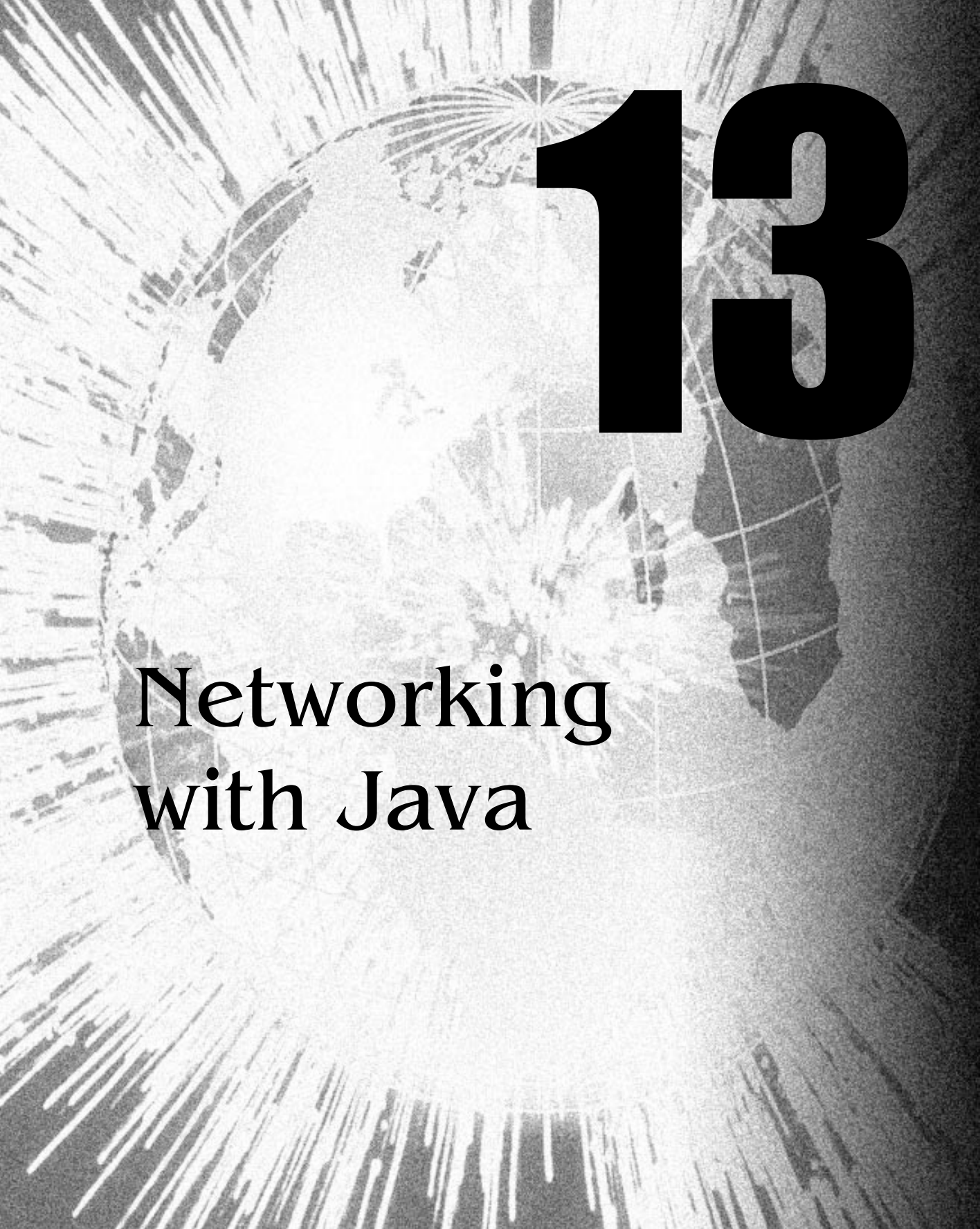
JAVA

PROGRAMMING LANGUAGE HANDBOOK

Anthony Potts

David H. Friedel, Jr.

CORIOLIS GROUP BOOKS



13

Networking with Java

13 Networking with Java

Java supports a number of classes and methods so that you can make connections to the Internet and perform a variety of operations from processing Web content to transferring files.

Since you are reading this book, it's likely that you are connected to the Internet in one way or another. And after writing a few Java applets, you're probably aware that networking plays a key role in the development of Java. So, whether you are just stepping into the world of networking or you already realize what a pain networking can be, this chapter should help you use Java to connect your applications to other resources via networks.

We'll begin by examining the basics of how networking works across the Internet. As you'll see, the two key networking components for the Internet are protocols and ports. We'll then show you how to use special classes defined in the java.net package to set up network sockets and support client/server network communications. We'll also show you how to use the URL class to access content from the Web. In the last part of the chapter, we'll look at techniques for communicating between applets.

Understanding the Basics of Networking

You may not be aware of it, but you already know the basics of computer networking, especially if you use a telephone. A telephone provides a means for two or more people at different locations to talk back and forth. In order for voices (data) to be sent back and forth, a *protocol* is used that defines how each party

will interact with each other. In the case of a telephone, the protocol involves speaking into a transmitter and listening to a receiver.

When it comes to computer networks that connect with the Internet, the set of key protocols that are used include:

- TCP/IP
- SMTP
- FTP
- HTTP
- NNTP
- Finger
- WhoIs

Let's take a few moments and discuss the basics of each of these protocols.

TCP/IP

The full name of this protocol is *Transmission Control Protocol/Internet Protocol*. It was proposed as a standard in 1973 but was not adopted as a standard until 1982. TCP/IP serves as a hardware-independent protocol for allowing different types of networking systems to connect and send data back and forth. Actually, TCP/IP consists of a collection of different protocols.

The Unix operating system was the first major OS to incorporate this protocol for network communications. With the proliferation of the Internet, this protocol has become widely used for both Internet connections and for networking computers in local area networks. To take advantage of Java's networking capabilities, you'll need to have a TCP/IP connection to the Internet. Fortunately, newer PC operating systems, such as Windows NT, are incorporating TCP/IP into the system. If you are using an older operating system like Windows 3.1, you'll need to make sure TCP/IP has been set up properly on your computer.

SMTP

SMTP stands for *Simple Mail Transfer Protocol*. It is the protocol that is used by email programs such as Eudora to send and receive email. The communications work of handling email is actually controlled by three protocols: SMTP, POP3, and MIME. The primary protocol used to process email is SMTP.

FTP

FTP stands for *File Transfer Protocol*—the protocol that allows files to be transferred from one computer to another across the Internet. This protocol is the most widely used, since most Internet users spend a majority of their time transferring files. Programs that automate the process of sending and receiving files for you use this protocol behind the scenes to make sure that all data is transferred correctly.

HTTP

HTTP stands for *Hyper Text Transfer Protocol*—the protocol that drives the World Wide Web. Since you've probably written countless HTML documents, you already know that information is processed on the Web in the form of hypertext links. HTTP ensures that Web content is transferred correctly from a server to a client computer so that information can be displayed in the correct format. Web browsers like Netscape live and die by the HTTP protocol. You've probably used this protocol numerous times since Web addresses are usually specified by first including the HTTP protocol (for example, http://www.coriolis.com).

NNTP

NNTP stands for *Network News Transfer Protocol*. If you have ever been involved with an Internet newsgroup, you've probably used this protocol hundreds of times, without ever knowing it existed. NNTP works behind the scenes to deliver messages from newsgroups that you belong to.

Finger

Imagine having a finger so large that you could reach out and touch anyone who is connected to the Internet to see if they are currently available. This might sound a little ridiculous, but this is the goal of the Finger protocol. With this protocol, you can "finger" an Internet user to obtain information that the user has declared as public. Some of this information might include the user's full name, mailing address, phone number, and so on.

WhoIs

If you live in a big city and you want to locate someone, you can go to a phone book and look the person up. On the Internet, you can use the WhoIs protocol to do your detective work for you. As long as a person, organization, or company is registered with the Internet address assignment agency (InterNIC), this protocol will give you the information you need.

The Client/Server Model

Using our simple phone analogy, let's investigate the concepts of client/server technology—the driving force that makes networks like the Internet come to life. At one time or another, we've all stood by the phone in anticipation waiting for a call to come in. The caller has the option to pick up the phone at any time and make the call. When they do, we can pick up the phone after a few rings to complete the connection. Since we are the party who is waiting for the call, we operate as the *server*. The person making the call, on the other hand, operates as the *client*, since by making the call, he or she is simply "requesting information" from us—the server.

Computers connected in a network pass information back and forth in the form of *messages*. The computer, acting as the client, sends messages to the server. The server processes the messages and returns information in the form of messages back to the client.

Ports of Interest

In addition to performing complex calculations quickly, today's computers are great at performing multiple tasks at the same time. (Or at least they create the illusion that they can, as we learned in Chapter 8 when we explored threads.) Computers connected to the Internet as servers can provide many services at once. They do this by using *ports* that link up different channels set up at a single address. A computer may set up a port with a number between 0 and 65535, known as a 16-bit number. Each request that comes in from a client requires a port specification as shown in Figure 13.1. If the server has a service running for that port, it is addressed with a certain message defined by the protocol. Currently, ports have been established for the default protocols. For example, port 80 is reserved for HTTP, port 21 is reserved for FTP, port 119 is reserved for News, and so on. The set of key protocols and their associated ports are listed in Table 13.1.

Table 13.1 Key Internet Protocols and Their Associated Ports

Protocol	Port
HTTP	80
FTP	21
NNTP	119
Whols	43
SMTP	25
Finger	79

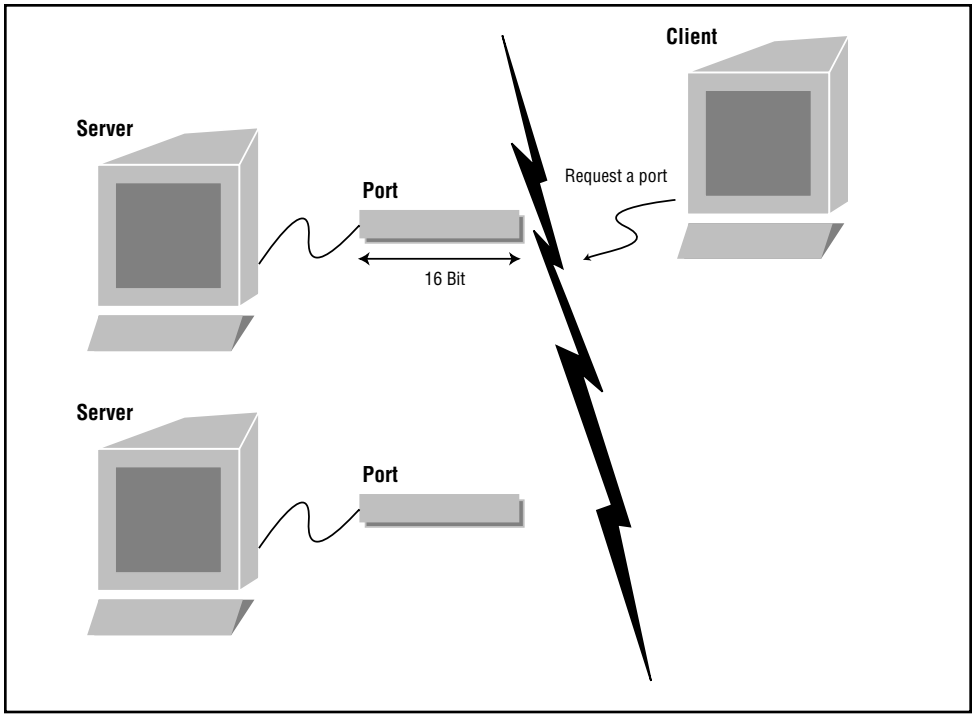


Figure 13.1
Processing a request from a client.

Establishing New Ports and Protocols

You can create your own ports and protocols for the Internet—in fact new ones are being established for the Internet all the time. Just remember that the basic principal behind a network is to be able to communicate; thus, standards are needed. To make sure that everyone follows standard practices for setting up communication links on the Internet, a system of *Request for Comments* (RFCs) have been established. RFCs are documents that are submitted for the purpose of defining specifications or the guidelines for the Internet. For example, the FTP protocol was submitted and reviewed by many Internet developers and users until it was officially adapted as RCF0959. This particular document presents all of the specifications for the protocol. If you were to create a protocol, you could submit it for review and it to would go through the process of being reviewed. Think

of it as a bill on Capitol Hill on its windy path to becoming a law. There are many chances for it to be vetoed along the way.

Introducing the java.net Package

As we've explored topics like creating GUI interfaces, controlling threads, and writing exceptions, we've seen that Java's developers put a lot of care into creating packages and classes to handle important activities. Fortunately, when it comes to networking, the Java developers came through again. The java.net package provides several classes for handling key Internet protocols. With these classes, you can perform a number of Internet related operations, including transferring files, setting up and controlling mail servers, creating search agents, processing Web content, and much more.

The structure of the java.net package is shown in Figure 13.2. Table 13.2 provides a summary description of each key class. To understand how these key

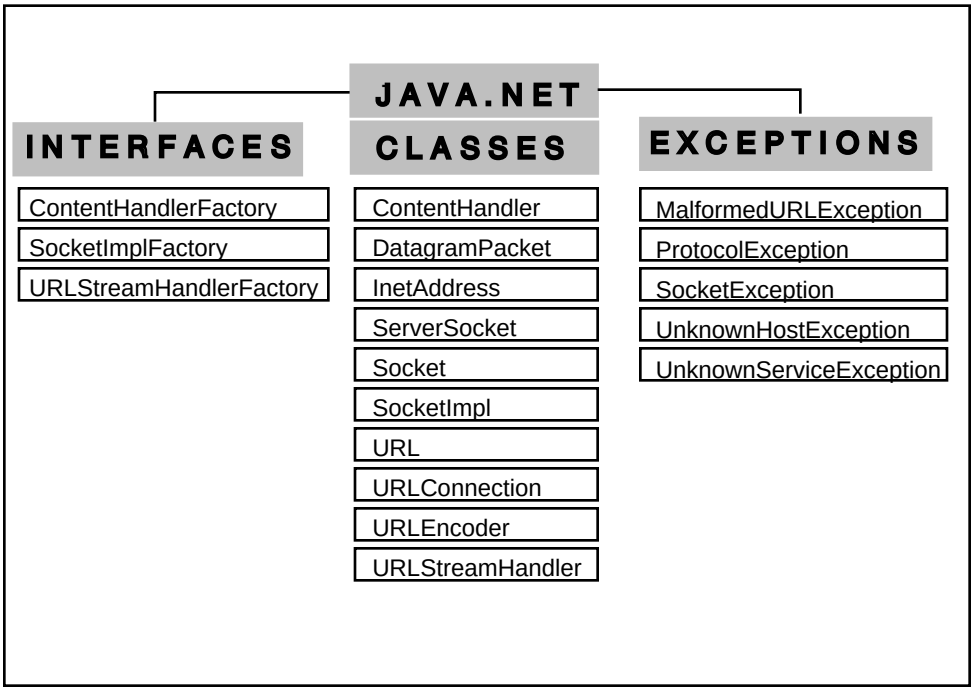


Figure 13.2

The structure of the java.net package.

Table 13.2 The Main Classes in the java.net Package

Class	Description
ContentHandler	This is an abstract class that processes data in a specific format from a stream and creates an appropriate object.
InetAddress	Used to process an Internet address.
ServerSocket	Used to set up a connection-oriented server socket.
Socket	Used to set up a socket for a client.
SocketImpl	Used to create a specific implementation of a socket.
URL	Used to set up a URL.
URLConnection	Used to set up a connection to a URL.
URLStreamHandler	Used to manage data read from or written to a URL.

classes operate, you'll need to be familiar with the client/server model of networking and how sockets are used to facilitate network communications. Because of their complexity and depth, we won't cover every class in detail; but we will cover the essentials so that you can use some of these classes to develop network-capable Java applets and applications.

Networking Applications

Although you may only be creating Java applets right now, opportunities will emerge for creating more powerful distributed programs, especially as Java compilers get better at generating more tightly compiled code. The incredible flexibility and power that Java's portability provides across multiple platforms allows you to create programs for wide distribution. As you gain more experience developing with Java, you'll find yourself building applications that perform networking related tasks that were never possible in the past using traditional languages like C and C++. The key is learning how to exploit the client/server model of developing networked applications.

Working with Internet Addressing

Before we jump in and start using some of the java.net classes to set up client/server communications, you'll need to know the basics of TCP/IP Internet addressing. Actual Internet addresses are represented using a series of four numbers separated by dots. Here's an example:

```
189.12.49.123
```

Each number must be in the range from 0 to 255.

Since you should already have your own Internet address, you can easily determine what it is by using the **InetAddress** class defined in the java.net package. This class provides five key methods as listed in Table 13.3.

To see how the **InetAddress** class can be used, let's write a simple Java program that checks your local computer and displays your Internet address:

```
import java.net.*;  // Must be included to use the InetAddress class

public class checkMyIP {
  public static void main(String args[]) {
     InetAddress localA = null;
     try {
         // Returns the name of the local address
         localA = InetAddress.getLocalHost();
      }
      catch {
      }
      system.out.println("The local address is " + localA);
   }
}
```

First notice that the **import** statement is used to include the classes from the java.net package. The variable **localA** is declared as type **InetAddress**. A call to the method **getLocalHost**() returns the address of the local host computer. Notice that no parameters are provided.

Table 13.3 The Key Methods in the InetAddress Class

Method	Description
getHost()	Returns the name of the host being stored by the **InetAddress** class. An example would be *coriolis.com.*
getAddress()	Returns the numerical address of the host being stored by the **InetAddress** class.
getLocalHost()	Returns the name of the local host.
getByName()	Returns the name of the host specified by a supplied argument to this method.
getAllByName()	Returns a list of addresses for the host name provided as an argument to this method.

The Role of the Client

A client application interacts with other clients to perform certain functions through *sockets*. In order for a client to communicate to other clients, a server must be established for connection. The server is discussed later in the chapter, but for now all you need to know is that a client requires a server to make a connection from which all communication originates.

Creating a Socket

A socket provides a means for information to flow to and from a computer, much like the way a wall socket provides a channel for electricity to flow through. To initiate a socket, the client application must create an instance of the **Socket** class. This is a class that Java provides to facilitate Internet communications. It uses Internet protocol (IP) addresses and port numbers. The IP address indicates with which computer you want to establish communication. Here is an example of how a socket object is created using the **Socket** class:

```
Import java.net.*;
...
Socket mrClient = null;
```

First, notice that the package java.net must be imported so that you can access the **Socket** class. Here the socket object, **mrClient**, is declared and set to null. To initialize **mrClient**, you use a statement like the following:

```
mrClient = new Socket(165.247.88.17, 10);
```

This code creates an instance of the **Socket** class with an IP address of 165.247.88.17 and a port value of 10.

Using Sockets

After creating the gateway for data to enter your program, you need to create a repository for the data to come and go. Streams provide a method of reading and writing data to a socket. For our example and most uses, we'll need two separate streams. The first stream is an input stream and calls a method to receive the information from the socket. The second stream is used to send data out from the socket:

```
InputStream rawDataIn = mrClient.getInputStream();
OutputStream rawDataOut = mrClient.getOutputStream();
```

The first statement instantiates the object **rawDataIn** as type **InputStream**. Any data that flows to the client, **mrClient**, will be stored in **rawDataIn**. The other stream is the **OutputStream**. As you might have guessed, this stream is responsible for passing the data from the client. If you recall from Chapter 12, the Input/OutputStream is difficult to parse all data through because not all of the data processed is just simple bytes. To use some of the more complex data types available in Java, we need to pass our stream through the **DataInput** and **DataOutput** interfaces. This allows for more control over the data that is to be received and sent. (For more information on this technique, refer to Chapter 12.) To pass both the incoming and outgoing data through the **DataInput** and **DataOutput** interfaces, you'll need to use code like the following:

```
DataInputStream DataIn = new DataInputStream(rawDataIn);
DataOutputStream DataOut = new DataOutputStream(rawDataOut);
```

An InputStream/OutputStream is used because of the extensive ability to manipulate more complex streams than simple bytes. For example, you can test the stream to see if it is at the end of a line by using a **readLine**() method. This method explicitly checks for *returns, newlines,* and *return-newline* combinations in a string of characters read from the stream.

Creating a Sample Client Application

Now that you've been introduced to the basics of how client applications communicate using sockets and IP addresses, let's write a sample Java program that creates a socket object and reads in data:

```
import java.io.*;
import java.net.*;  // Must be included to use Socket class
import java.awt.*;

public class mrClient extends Frame {
   mrClient() { }
   mrClient(String s) {
      // Create the text display for the application
      super("Client Application");
      TextArea clientScreen = new TextArea("mrClient's Screen:\n", 10, 40);
      add("Center", clientScreen);
```

```
    pack();
    show();

    Socket mrClient = null;
    InputStream rawDataIn = null; // Setup the input and output streams
    OutputStream rawDataOut = null;

    try {
      mrClient = new Socket(s, 10 );   // Create the socket object

      rawDataIn = mrClient.getInputStream();
      rawDataOut = mrClient.getOutputStream();
      // Create a data input stream
      DataInputStream DataIn = new DataInputStream(rawDataIn);
      // Create a data output stream
      DataOutputStream DataOut = new DataOutputStream(rawDataOut);
      int Value = DataIn.readInt();  // Read in data from the socket
      clientScreen.appendText( "mrClient receives -  " + Value + "\n" );

      switch(Value) {   // Determine which value has been obtained
      // from the socket
       case 1:
         clientScreen.appendText("Your Number 1");
         DataOut.writeInt( 1 );
         break;
        case 2:
         clientScreen.appendText("Your Number 2");
         DataOut.writeInt( 2 );
         break;
        case 3:
         clientScreen.appendText("Your Shakey");
         DataOut.writeInt( 3 );
         break;
       case 4:
         clientScreen.appendText("Your a loser");
         DataOut.writeInt( 4 );
         break;
       case 5:
         clientScreen.appendText("This is my lucky number!");
         DataOut.writeInt( 5 );
         break;
       default:
         clientScreen.appendText("What was your Number?");
         DataOut.writeInt( 6 );
       }
       mrClient.close();   // Close the socket
}
```

```
// Handle exceptions that have occurred
catch( UnknownHostException e ) {
      clientScreen.appendText("Can't find Server");
    }
    catch( IOException e ) {
      clientScreen.appendText("an IO Error has occurred");
    }
  }

  public static void main( String args[] ) {
      new mrClient(args[0]);
    }

     // Handle events that have occurred
     public boolean handleEvent(Event evt)
     {
        switch(evt.id)
        {
           System.out.println(evt.target.toString());
           case Event.WINDOW_DESTROY:
           {
               System.out.println("Exiting...");
               System.exit(0);
               return true;
            }
         }
      return true;
      }

}
```

When you run this code, remember to pass the address of the computer to connect to. Otherwise, you will throw an **ArrayOutOfBoundsException**. In this case, you should pass the address of your local computer as the command line argument. A sample output produced by the program is shown in Figure 13.3.

Most of the interesting code is found in the **mrClient**() constructor. After setting up an introductory message, the constructor declares objects that will be used for creating a socket and the input and output streams:

```
Socket mrClient = null;
InputStream rawDataIn = null;    // Set up the input and output streams
OutputStream rawDataOut = null;
```

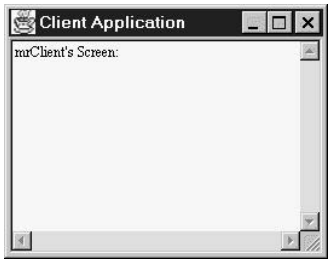


Figure 13.3
Output produced by the client application.

Next, notice the **try** statement that comes before the line of code that creates the socket object, **mrClient**:

```
try {
   mrClient = new Socket(s, 10);   // Create the socket object
...
```

Because all sockets, as well as streams, are capable of throwing **IOExceptions**, it is good practice to check for an IOException when creating a socket and using the socket to perform critical operations such as sending and receiving data. (For more information on exceptions, review Chapter 11.)

The next set of statements perform the actual reading and writing to the socket. In the line shown next, **DataIn** is the reference to the **DataInput** which is linked to the **InputStream** of the socket:

```
int Value = DataIn.readInt();
```

In this case, the integer **Value** receives the value of the **DataIn** at that instance and clears the buffer. There are other types of data that may be read from the stream these are discussed further in Chapter 12. Once the data has been read from the socket, we use a **switch** statement to check the input and display an appropriate message. After the input data has been processed, the **close**() method is called to close the socket.

Networking Concerns

When communicating across a network, certain factors should be taken into consideration. In a perfect world, all users would have a fast connection with no lag. Unfortunately that "perfect" world doesn't exist, so you must take the bandwidth of your connections into consideration. To optimize your applications, the use of return codes allows for quick transmission of integers or small clauses that have more complicated procedures attached to them on either side. In our example, the receiving and returning of integers actually fall into a **switch** statement that performs a more complicated task. The following line is responsible for sending the integer "5" to **DataOutputStream** interface:

```
DataOut.writeInt( 5 );
```

From here, the data is passed through the interface and converted to bytes for the OutputStream to be able to handle the data.

Bring on the Server

Now that we have the client under our belt, we need to learn how to connect to another application. This is accomplished by having the client connect to a server. Essentially, a server is a computer that provides a service for others to utilize. The server is set to listen to a particular port. Once a client requests information on that port, the server springs into action.

For setting up and controlling servers, Java provides a counterpart to the **Socket** class called the **ServerSocket** class. Objects created from this class do not connect to an address like socket objects do. Instead, a server socket object creates a port for another socket to plug in to. To declare a variable of the **ServerSocket** class, you use a statement like the following:

```
ServerSocket mrsServer = null;
```

Then, you can create an instance of the **ServerSocket** class by supplying the appropriate parameter to the class' constructor:

```
mrsServer = new ServerSocket( 10 );
```

This line creates an instance of a **ServerSocket** class named **mrsServer**. The object, **mrsServer**, is initialized with a port setting of 10. Essentially, the server is now set up to listen for incoming sockets at port 10. In addition to specifying the port, you can set a time limit. Here's an example:

```
mrsServer = new ServerSocket(10, 5);
```

Once you have created an object of type **ServerSocket**, you must put the server in motion by using the **accept**() method. This method effectively places a hold on the program until a socket comes along to connect to the server. So, it would be wise to place this method call in a thread if you want to perform other functions while you are waiting for a socket to connect to the server. Once a socket shows up, the **accept**() method returns a socket. Then, the server operates just like the client, by allowing sockets to communicate back and forth.

Creating a Sample Server Application

Let's combine everything we know about clients and servers and create a program that uses both the **ServerSocket** and **Socket** classes. This program will create a server socket object and then wait for a client object to connect up. Once the client connects, the server will try to read data from the client. Here's the complete program:

```
import java.io.*;
import java.net.*;
import java.awt.*;

public class mrsServer extends Frame {
   TextArea serverScreen;

    mrsServer() {
     super("Server Application");
     serverScreen = new TextArea("mrsServer's Screen:\n", 10, 40);
     add("Center", serverScreen);
     pack();
     show();

     ServerSocket mrsServer = null;   // Initialize the server socket
     Socket socketReturn = null;      // Initialize the client socket
     InputStream rawDataIn = null;
     OutputStream rawDataOut = null;
```

```
try {
    mrsServer = new ServerSocket( 10 );  // Create the server object
    socketReturn = mrsServer.accept();  // Wait for the client
                                        // socket to come in
    serverScreen.appendText( "Connected to the mrsServer \n" );

    rawDataIn = socketReturn.getInputStream();
    rawDataOut = socketReturn.getOutputStream();
    DataInputStream DataIn = new DataInputStream(rawDataIn);
    DataOutputStream DataOut = new DataOutputStream(rawDataOut);
    DataOut.write( 5 );
    int Value = DataIn.read();

    switch(Value)
    {
    case 1:
      serverScreen.appendText("You sent me Number 1");
      break;
    case 2:
      serverScreen.appendText("You sent me Number 2");
      break;
    case 3:
      serverScreen.appendText("You sent me Number 3");
      break;
    case 4:
      serverScreen.appendText("You sent me Number 4");
      break;
    case 5:
      serverScreen.appendText("You sent me Number 5");
      break;
    default:
      serverScreen.appendText("What was your Number?");
    }

    }
    catch( UnknownHostException e ) {
      clientScreen.appendText("Can't find Server");
    }
    catch( IOException e ) {
      clientScreen.appendText("an IO Error has occurred");
    }
 }

 public static void main( String args[] ) {
     new mrsServer();
 }
```

```
    // Handle events that have occurred
    public boolean handleEvent(Event evt)
    {
      switch(evt.id)
      {
        System.out.println(evt.target.toString());
        case Event.WINDOW_DESTROY:
        {
           System.out.println("Exiting...");
           System.exit(0);
           return true;
        }
      }
     return true;
     }
}
```

This program is essentially an extension of the client example we created earlier. A sample output is shown in Figure 13.4. This time around we added server capabilities by using the **ServerSocket** class. The key code is found in the **mrsServer**() constructor. As you start to read this code, you might want to compare it with the code found in the **mrClient**() constructor from the previous example.

The first two important lines in **mrsServer**() declare the variable **mrsServer** of type **ServerSocket** class and **socketReturn** of type **Socket**:

```
ServerSocket mrsServer = null;   // Initialize the server socket (server)
Socket socketReturn = null;      // Initialize the client socket
```

In addition to the **ServerSocket**, we must declare a socket that is returned when

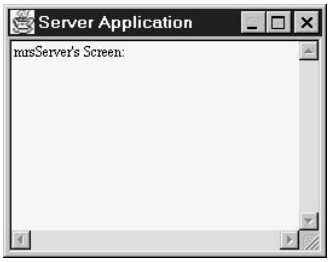


Figure 13.4
Output produced by the server application.

we connect to another socket. In the following lines of code, we create an instance of the **ServerSocket** class and initialize the port to 10. Then, the **accept**() method is called to instruct the program to wait for the incoming socket. The returning socket will be assigned to **socketReturn**:

```
try {
   mrsServer = new ServerSocket( 10 );  // Create the server object
   socketReturn = mrsServer.accept();   // Wait for the client socket to come in
```

In the event of a connection, the server will write a "5" to the socket stream and not to the server socket stream. After this occurs, the server will attempt to read data from the client, and the data will be processed by the server by displaying an appropriate message.

Web Content Only Please

If communicating across the Internet to access Web content is your only concern, Java has a few special classes for you—**URL** and **URLConnection**. You can use these classes for creating objects to read, manipulate, and write Web content. Here is the syntax you use to declare a Uniform Resource Locator (**URL**) variable:

```
URL aURL = null;
```

In this case, **aURL** is the name of the variable being declared. To create and initialize an object you would use a statement like this:

```
aURL = new URL(http, www.coriolis.com, 80, index.html);
```

This line passes the **URL** class four parameters including the Internet protocol, Web address, Internet port, and the name of an HTML file to access. Although we are using four arguments with this constructor call, other variations are available including:

```
URL(String protocol, String host, String file);
URL(URL context, String spec);
URL(String spec);
```

Since the **URL**() constructor is called to create an object based upon a value, you should use **try...catch** statements to check for errors that might occur. This will allow you to check for a **MalformedURLException** that could be thrown when-

ever a URL is not specified properly.

Once a URL object has been created, you can easily use it to access Web content. Let's look at an example that grabs a file and prints it to the screen:

```
import java.io.*;
import java.net.*;

public class GrabAFile {
    BufferedInputStream holdingRoom1;
    String holdingRoom2;

    GrabAFile() {
    }

    GrabAFile( String userURL ) {
        URL aURL;
        InputStream rawDataIn;
        DataInputStream refinedDataIn;
        int aCharDataIn;

        try {
            aURL = new URL(userURL);
            rawDataIn = aURL.openStream();
            holdingRoom1 = new BufferedInputStream( rawDataIn );
            refinedDataIn = new DataInputStream( holdingRoom1 );

            while((aCharDataIn = refinedDataIn.read() ) >= 0 ) {
                    System.out.print((char)aCharDataIn);
            }

            System.out.print( refinedDataIn );
        }
         catch( MalformedURLException e ) {}
         catch( IOException e ) {}
         catch( Exception e ) {}
    }

    public static void main(String args[]) {
        new GrabAFile(args[0]);
    }
}
```

First, this code declares and initializes a URL object as explained earlier. Then, we open a stream for the input. To do this we declare an object as type **InputStream** and assign it a variable. Then, we make reference to the method

openStream() using the name of the object of type URL we wish to connect to the input stream.

Using the URLConnection Class

Another class worth mentioning when reading and writing data to a URL is the **URLConnection** class. This class is an abstract class and must be subclassed to implement the methods that are contained within it. It offers an abundance of useful methods for extracting data from files and setting up properties for writing to a URL.

To utilize any of the methods available in the **URLConnection** class, you first declare a variable of type **URLConnection** as shown here:

```
URLConnection anotherURL;
```

Next, you need to open the connection to an instance of an existing URL. To accomplish this task, you would use a statement like this:

```
anotherURL = aURL.openConnection();
```

The variable, **anotherURL**, is declared as type **URLConnection** as shown in the previous statement. The variable **aURL** in the call, **aURL.openConnection**(), is an object instantiated as type **URL**. The method **openConnection**() creates an object of type **URLConnection** and establishes a connection between the remote object declared in the URL. Let's look at an example where we extract the content type of the file being accessed:

```
import java.net.*;

public class GrabInfo {

    GrabInfo() {
    }

    GrabInfo( String userURL ) {
        URL aURL;
        URLConnection anotherURL;
```

```
        try {
            aURL = new URL(userURL);
            anotherURL = aURL.openConnection();
            anotherURL.connect();
            System.out.print(anotherURL.getContentType() + "\n\n");
        } catch( MalformedURLException e ) {
        } catch( Exception e ) {
        }
    }

    public static void main(String args[]) {
        new GrabInfo(args[0]);
    }
}
```

Here we call the method **getContentType**() defined in the **URLConnection** class and print out what was returned from the object. This method can be extremely useful for determining file types. If you need a more thorough check of a file type, try the method **guessContentTypeFromStream**() and apply it to the previous example by passing the stream to it.

Networking between Applets

Situations may arise where you want applets on the same Web page to be able to communicate with each other. For example, if you want a certain event to occur in an applet when another applet reaches a point in an animation, you could have the animation applet send a message to the other applet at the appropriate time. The only limitation for this type of communication is that both applets must be present on a page to communicate back and forth. Currently, there is no way to have one applet send information to another applet on another page. Hopefully, this feature will be added in the near future.

Passing Information to a Fellow Applet

Let's write some Java code that allows one applet to communicate with another. After you compile and run the code, you should see the responses shown in Figures 13.5 through 13.7. One of the applets is named Gentleman and the other is named Lady.

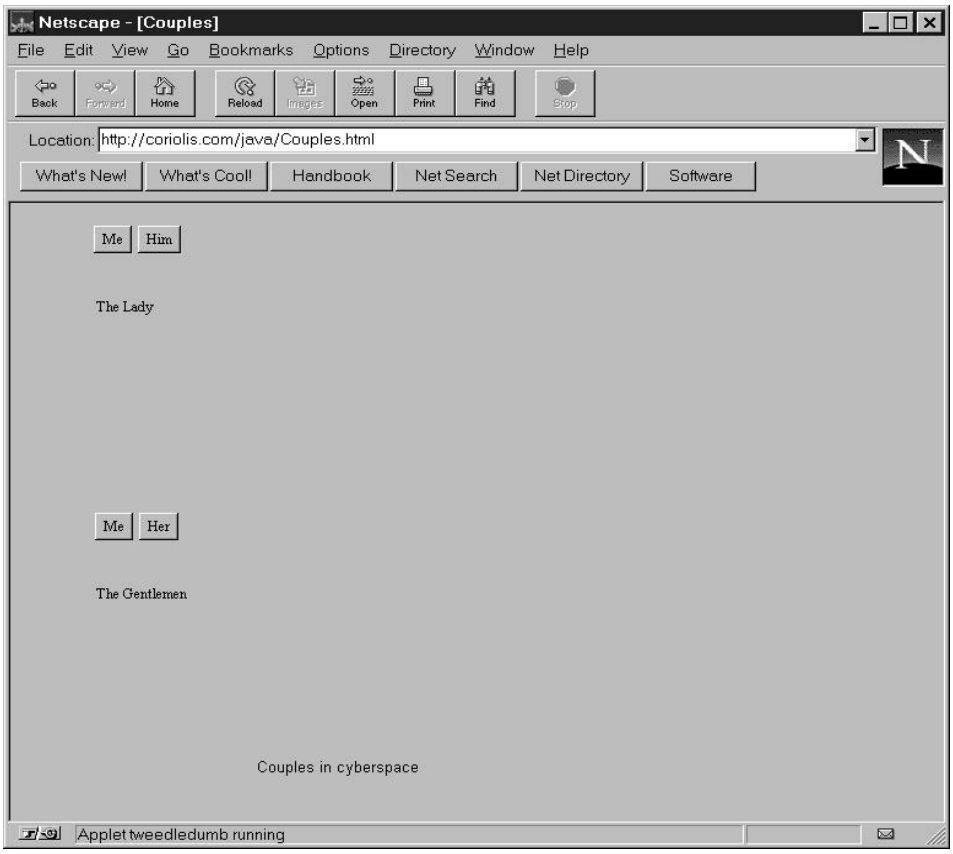

Netscape - [Couples]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://coriolis.com/java/Couples.html
What's New! What's Cool! Handbook Net Search Net Directory Software
Me Him
The Lady
Me Her
The Gentlemen
Couples in cyberspace
Applet tweedledumb running

Figure 13.5

A shot of the opening screen showing the two applets.

This example can quickly be modified to perform even more complex tasks with little or no effort. But first, let's discuss exactly what is taking place. The first applet, the Lady, looks like any other applet except for the **handleEvent**() method:

```
public boolean handleEvent(Event evt)
{
  if (evt.target instanceof Button) {
     if("Him".equals(evt.arg))
     {
       Gentleman tweedledee=
         (Gentleman)getAppletContext().getApplet("tweedledee");
       if ( tweedledee != null) {
         return  tweedledee.handleEvent(evt);
       } else {
```

```
        return false;
     }
   }
   else if("Me".equals(evt.arg))
   {
        str ="You Clicked!!";
        repaint();
        return true;
     }
     else if("Her".equals(evt.arg))
     {
        str ="Hello Mister!!";
        repaint();
        return true;
     }
```

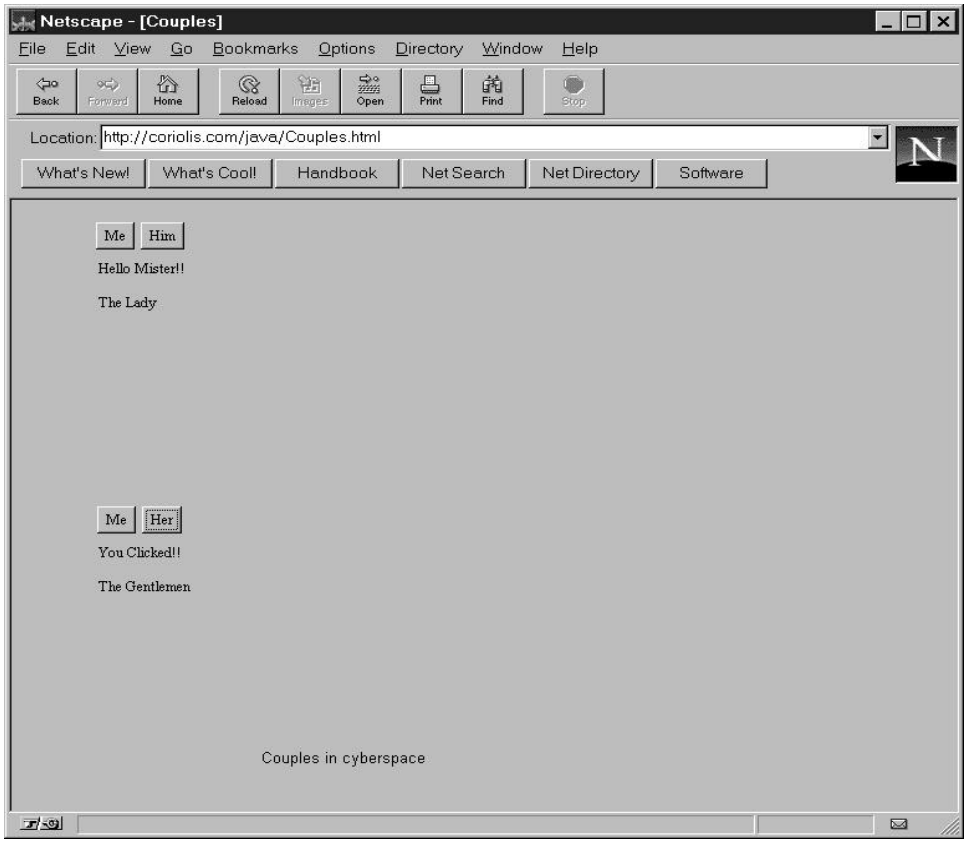

Netscape - [Couples]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://coriolis.com/java/Couples.html
What's New! What's Cool! Handbook Net Search Net Directory Software
Me Him
Hello Mister!!
The Lady
Me Her
You Clicked!!
The Gentlemen
Couples in cyberspace

Figure 13.6

A shot of the Gentleman applet's Me button pressed and the Her button pressed.

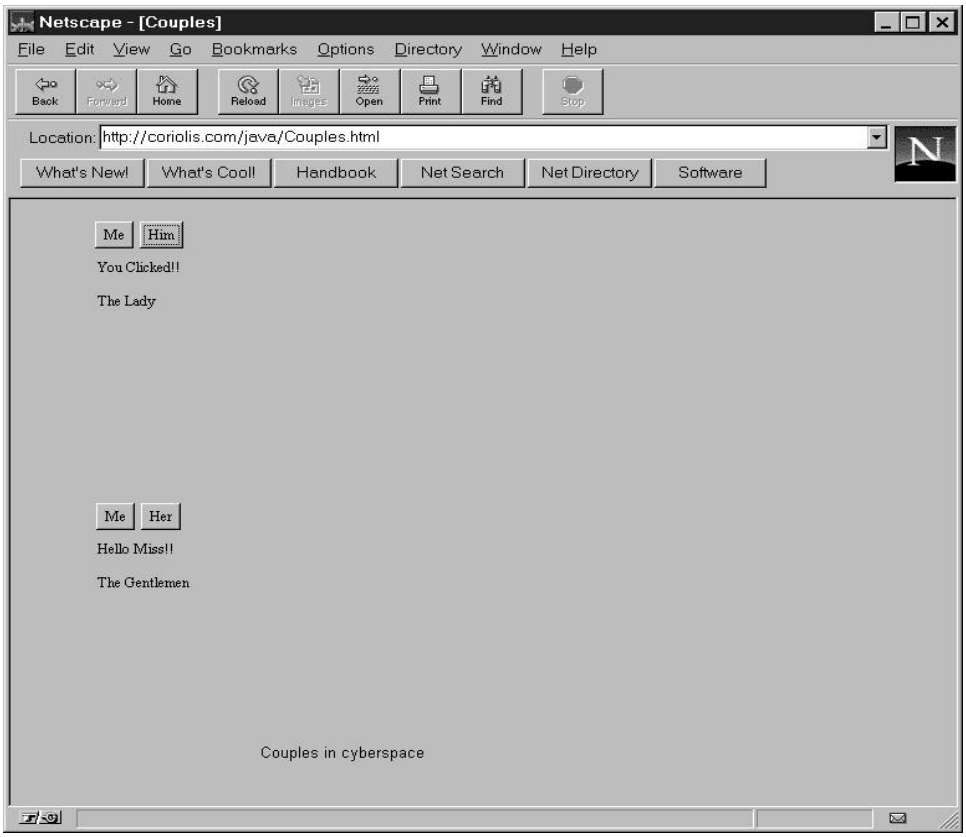

Netscape - [Couples]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://coriolis.com/java/Couples.html
What's New! What's Cool! Handbook Net Search Net Directory Software
Me Him
You Clicked!!
The Lady
Me Her
Hello Miss!!
The Gentlemen
Couples in cyberspace

Figure 13.7
A shot of the Lady applet's Me button pressed and the Him button pressed.

```
        return super.handleEvent(evt);
}
return super.handleEvent(evt);
}
```

The actual communication code is found in the second **if** statement. The first line actually retrieves the applet named tweedledee. This applet is named within the declaration of an applet in the HTML file named Couples:

```
<applet code="Gentleman" name=tweedledee width=187 height=232></applet>
```

Using the methods **getApplet**() from the **AppletContext** class and the **getAppletContext**() from the **Applet** class, we create an interface in which to

call the methods of the **Gentleman** class and apply it to the applet itself. This allows us to control the applet by calling it with the name of the variable and the name in the **<applet>** tag assigned to it. In this case, **tweedledee** controls the applet from within the **Lady**.

Now we need to make sure when calling handling events that we make reference to the particular event:

```
// From the Lady
return  tweedledee.handleEvent(evt);

// From the Gentleman
else if("Him".equals(evt.arg))
   {
     str ="Hello Miss!!";
     repaint();
     return true;
   }
```

In the event that a user clicks the "Him" button from the "Lady" applet, the Gentleman must address the event that will be passed to it. Otherwise, the **evt** object will slip by without being acted upon. The same goes for the Lady addressing the events that will occur in the Gentleman. Everything here is just reversed for the other applet to communicate in the reverse direction.

Which Came First ... the Chicken or the Egg

When compiling the two source files for our applets listed at the end of this chapter, it is important to remember that both source files must be typed in and saved before compiling one or the other. This is because both reference each other's class file. When no class file is present, as in the event of requiring a class in both directions, the Java compiler reads the source file and compiles it from the definitions within. This would be comparable to a deadlock when relating this situation to threads.

The complete code for the Lady applet is:

```
import java.awt.*;
import java.applet.*;
```

```
public class Lady extends Applet
{
     String str = "";

   public void init()
   {
     Button a = new Button("Me");
     Button b = new Button("Him");
       add("East", a);
       add("West", b);

   }

public boolean handleEvent(Event evt)
{
  if (evt.target instanceof Button) {
     if("Him".equals(evt.arg))
     {
Gentleman tweedledee=
(Gentleman)getAppletContext().getApplet("tweedledee");
       if ( tweedledee != null) {
         return  tweedledee.handleEvent(evt);
       } else {
         return false;
       }
     }
     else if("Me".equals(evt.arg))
     {
        str ="You Clicked!!";
        repaint();
        return true;
     }
     else if("Her".equals(evt.arg))
     {
        str ="Hello Mister!!";
        repaint();
        return true;
     }
     return super.handleEvent(evt);
}
return super.handleEvent(evt);
}
     public void paint(Graphics g) {
```

```
        g.drawString(str, 60,50);
        g.drawString("The Lady", 60, 80);
     }
}
```

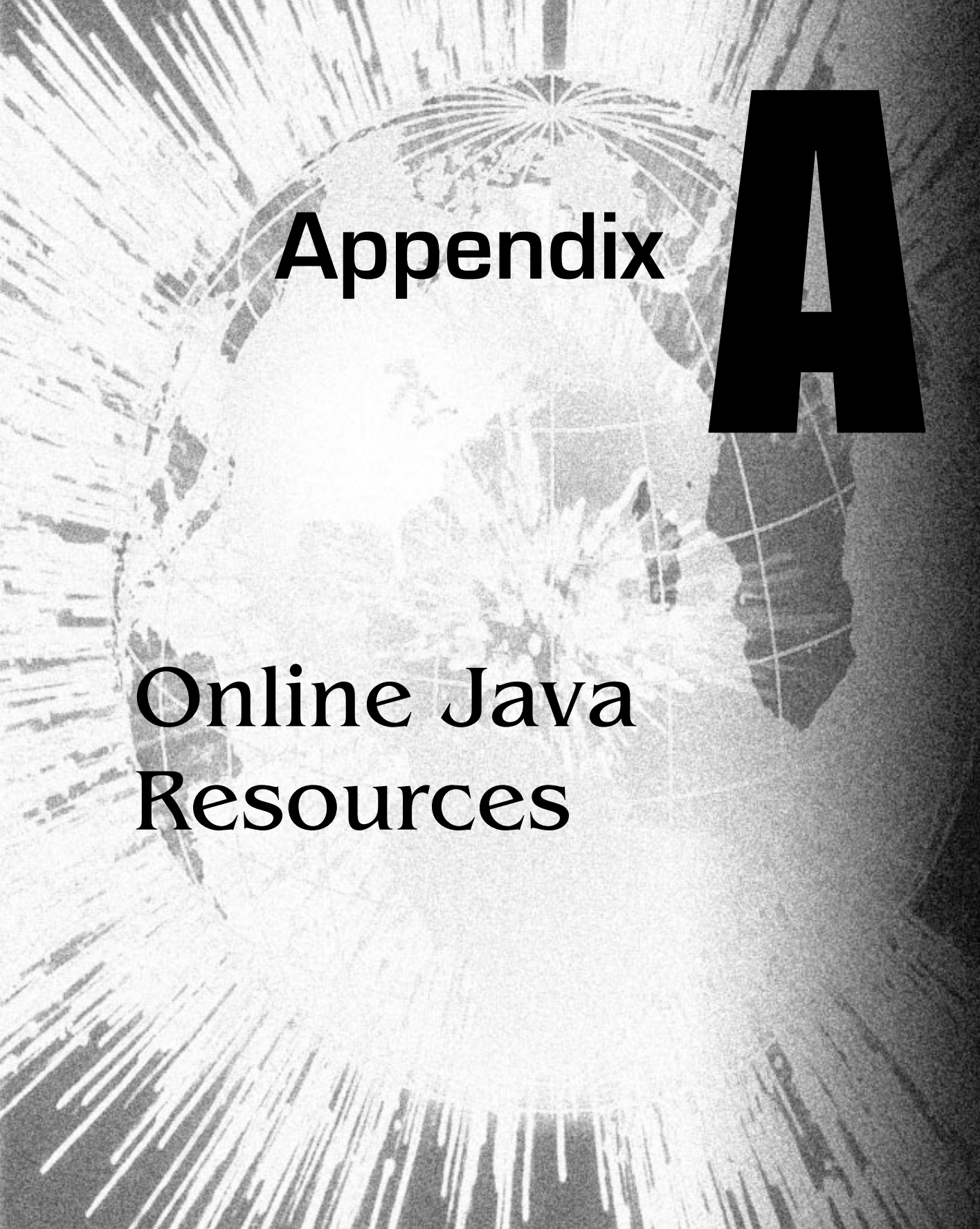
And here is the complete code for the Gentleman applet:

```
import java.awt.*;
import java.applet.*;

public class Gentleman extends Applet
{
     String str = "";

   public void init()
   {
     Button a = new Button("Me");
     Button b = new Button("Her");
       add("East", a);
       add("West", b);

   }

public boolean handleEvent(Event evt)
{
  if (evt.target instanceof Button) {
     if("Her".equals(evt.arg))
     {
       Lady tweedledumb= (Lady)
         getAppletContext().getApplet("tweedledumb");
       if ( tweedledumb != null) {
         return tweedledumb.handleEvent(evt);
       } else {
         return false;
       }
     }
     else if("Me".equals(evt.arg))
     {
         str ="You Clicked!!";
         repaint();
         return true;
     }
     else if("Him".equals(evt.arg))
     {
         str ="Hello Miss!!";
```

```
          repaint();
          return true;
        }
        return super.handleEvent(evt);
}
return super.handleEvent(evt);
}
   public void paint(Graphics g) {
        g.drawString(str, 60,50);
        g.drawString("The Gentleman", 60, 80);
     }
}
```

The Couples.html File

```
<HTML>
<HEAD>
<TITLE>Couples</TITLE>
</HEAD>
<BODY>
<applet code="Lady" name=tweedledumb width=187 height=232></applet>
<br>
<br>
<applet code="Gentlemen" name=tweedledee width=187 height=232></applet>
Couples in cyberspace
</BODY>
</HTML>
```



Appendix A

Online Java Resources

Online Java Resources

The amount of Java information available on the Web is incredible. You can find source code, applets, documentation, white papers, and many other useful resources. This appendix provides a description of all the best stuff we could find at press time. Make sure you use your favorite Internet search engines to keep looking for new resources. Send us an email if you find something that is not on this list, or if you have a site with an applet on it. We would love to see what you are working on and learn how this book has helped you to create Java applications and applets.

Java Programming Tools

Site: Javasoft - Sun's main Java site
Location: http://www.javasoft.com
Description: This is the best site to look for official information, documentation, and tools. From this site, you can download the Java Developer's Kit, check out the online documentation, or get licensing information. The applets aren't exactly earth-shattering, but this site is the best place to go to get *accurate* information from the creators of Java.

Site: Javamaker
Location: http://net.info.samsung.co.kr/~hcchoi/javamaker.html
Description: Javamaker is a neat little programming environment. It isn't extremely refined, but it is useful for programmers who prefer to use command-

Netscape - [JavaMaker - Simple Java Programming Environment]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://net.info.samsung.co.kr/~hcchoi/javamaker.html
JAVAMAKER (c)
Simple Java Applet Programming Environment!
Simple means Power!
JavaMaker is a Java Applet Programming Environment.
It provides easy way to edit, compile, and run Java Applets in Win32 environment.
Note: You must install a Win32 JDK before using JavaMaker.
You can get a copy of it from JDK 1.0 Beta Site.
Installation Guide
Document: Done

Figure A.1
The Javamaker home page.

line programming environments over some of the new visual environments. Figure A.1 shows the Javamaker home page.

Site: Diva
Location: http://www.inch.com/~friskel/rant.htm
Description: Diva is a visual design environment for Java. For being out so fast, It provides some impressive features. Diva will let you layout buttons and panels and a few other controls. Then, it automatically builds Java code for you, including event handling methods. Although Diva is not extremely robust at this time, it can help you create bigger applications in less time, especially when you are working with the AWT. The one drawback with Diva is that it requires that you use precisely laid out controls, which limits you from using the Layout Managers.

Site: Black Star's Link
Location: http://www.blackstar.com
Description: Black Star has taken a very creative approach to Java development. They have set up a CGI link that allows files on a user's local system to be compiled on their site. This is great for programmers who are working with operating systems that do currently support a Java compiler.

Site: Roaster Home Page
Location: http://www.natural.com/pages/products/roaster/flyer.html
Description: Roaster is the first Java compiler available for the Mac. It was just announced as this book was going to press, so we did not have time to fully test it out. Because of the popularity of Java, you can expect to see many more development tools for Mac programmers in the near future.

Site: Symantec's Java Central
Location: http://cafe.symantec.com
Description: Café is one of the first commercial Java development packages. It is an add-on to Symantec's C++ system. If you like Symantec C++, you will like Café. A Mac version of Café, called Caffeine is also in the works.

Site: Kalimantan Compiler
Location: http://www.dstc.edu.au/projects/kalimantan/
Description: Kalimantan is a collection of basic tools designed to simplify the task of constructing Java applications and applets. Kalimantan builds on the Sun-supplied Java API with a graphically-oriented interface. Kalimantan works under Solaris 2.4 and up, and Windows '95.

Kalimantan is bundled with the teikade (formerly known as dejava) tools from PFU Limited. We strongly suggest that you use both tools since each neatly complements the other, together forming a fairly complete programming environment.

The environment is inspired by the Smalltalk programming environment but tailored to specifically support programming in Java.

Archival Sites

Site: Gamelan
Location: http://www.gamelan.com
Description: If you could only visit a single site to get all your Java information, Gamelan should be it! That may sound a little extreme, but this site, as shown in Figure A.2, is both full of content and a joy to use. It is one of the few Web sites that has been able to use frames and applets in a way that is not overbearing. The site currently boasts over two thousand links to applets, tools, and source code! The resource links are categorized and are based on a very intuitive hierarchy. This is also an excellent place to submit your own applets for others to see and hopefully admire.

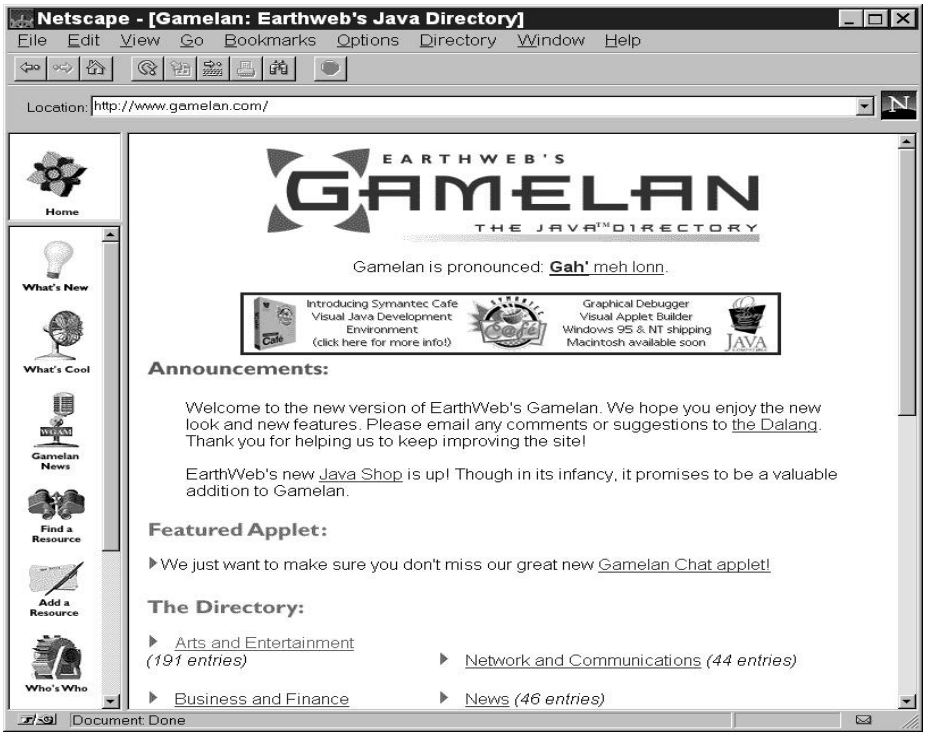

Netscape - [Gamelan: Earthweb's Java Directory]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://www.gamelan.com/
Home
What's New
What's Cool
Gamelan News
Find a Resource
Add a Resource
Who's Who
EARTHWEB'S
GAMELAN
THE JAVA™ DIRECTORY
Gamelan is pronounced: Gah' meh lonn.
Introducing Symantec Cafe Visual Java Development Environment (click here for more info!)
Graphical Debugger Visual Applet Builder Windows 95 & NT shipping Macintosh available soon
Announcements:
Welcome to the new version of EarthWeb's Gamelan. We hope you enjoy the new look and new features. Please email any comments or suggestions to the Dalang. Thank you for helping us to keep improving the site!
EarthWeb's new Java Shop is up! Though in its infancy, it promises to be a valuable addition to Gamelan.
Featured Applet:
We just want to make sure you don't miss our great new Gamelan Chat applet!
The Directory:
Arts and Entertainment (191 entries)
Network and Communications (44 entries)
Business and Finance
News (46 entries)
Document: Done

Figure A.2
Gamelan's Java applet resources.

Site: AltaVista Java Searching
Location: http://altavista.digital.com/
Description: Alta Vista is a killer search engine that lets you search for Java applets by name. The search engine has catalogued thousands of Java enabled pages and it parses out the name of the applets for you to view.

Site: Jump on Java
Location: http://www.coriolis.com/java
Description: This site was built by the authors of this book to feature both applets created by other developers and information about Java-related Coriolis Group books, including sample chapters and technical notes. We run a contest with prizes given out each week for the best applet. In addition, you'll find links to the best Java resources and development tools. Visit the site and let us know what you think. And when you create your next killer applet, submit it to the contest!

Netscape - [The Coriolis Group presents - "Get a Jump on Java" Web Sit...
File Edit View Go Bookmarks Options Directory Window Help
Location: http://165.247.200.139/java/index.html
The Coriolis Group Presents:
Get a Jump on Java
Java Books
Applet Gallery
Applet Contest
Online Resources
Enhanced Java Web Site
Get a Jump on Java!
Java is changing the face of the Web, making live, interactive online applications an incredible reality. The Coriolis Group now provides Java enthusiasts the best, most accessible Java information available: a new series of books that give them what they need to master the new language, and the "Get a Jump on Java" Web site, full of information not found anywhere else.
New Book on Java Available Now!
• Java Programming EXplorer - Authors: Neil Bartlett, Alex Leslie, and Steve Simkin; Availability: NOW!
Another Book on Java Coming Soon!
http://www.coriolis.com

Figure A.3

The Coriolis Group's Jump on Java Web site.

Site: JARS—Java Applet Rating Service
Location: http://www.jars.com
Description: This site, as shown in Figure A.4, has based its existence on becoming the defacto rating service for Java applets. So far, they have done pretty well. You will see many sites with the JARS logo and a "Top 5% Java Applet" sign.

Site: The Java Boutique
Location: http://weber.u.washington.edu/~jgurney/java/
Description: This is a great site to look for information about using applets on your Web page. It does not really cover Java programming techniques, but from a pure implementation point of view, it is a "must-see" site.

Netscape - [JARS]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://www.jars.com/
JARS
Java™ Applet Rating Service
JARS/JavaWorld Reviews
[Text-only JARS | Normal JARS | Framed JARS]
INFO About JARS, Logos, Rating System, Judges, Sponsors
NEW New Applets, Site Changes
RATINGS Top 1%, Top 5%, Top 25%, Top 10, Top 100, Rated
FEATURES Applet Bank, Jars-NetForum, Judge's Pick
SUBMIT Submit Applet, Judges Application
SEARCH Search the JARS Site
CONTACT Contact JARS
Copyright(c) 1995, 1996 WEB CREATIONS, Inc. All Rights Reserved
JARS, AppletPerfect, Top Web Applet, Best Web Applet, Top 5% Web Applet, are service marks of WEB CREATIONS, Inc.
All other trade and service marks are those of their respective owners. Java and HotJava are trademarks of Sun Microsystems, Inc., and refer to Sun's Java programming language and HotJava browser technologies. Web Creations/JARS is not sponsored by or affiliated with Sun Microsystems, Inc.
Document Done

Figure A.4
The Java Applet Rating Service (JARS).

Site: Chilly's Java Collection
Location: http://cecil.cac.psu.edu/~chilly/applets/
Description: This is just a fun little site with some good links.

Cool Stuff

Site: Café Del Sol
Location: http://www.xm.com/cafe/
Description: We could write a description of this site shown in Figure A.5 but it would not sound as near as good as their own, so here it is: "Deep within the bowels of Sun Microsystems, Inc. lies a lab... the New Media Marketing lab. In addition to many other responsibilities the members of this lab are charged with the task of developing Java programs. These programs, innocently known as applets, are occasionally released... bringing much joy and happiness to the virtual world. Those with a keen eye can spot these applets on Web pages and in CDs released by the developers within New Media Marketing lab.

The Café del Sol is the first bit of the 'outside world' the applets see. Here they congregate, compile amongst themselves, and finally (with the permission of their creators) venture out amongst the other Kilo-Mega-Gigabytes that whiz by the Café del Sol, Sun Microsystems, and every other place on the Internet. But here they reside first; here in their primal stages. If you are fortunate enough to have found the Café del Sol, then you are welcome to look around, investigate the applets, and basically get your fill of Java."

Site: Netscape's Java Applet Page
Location: http://home.mcom.com/comprod/products/navigator/version_2.0/java_applets/index.html
Description: What can we tell you about this site except that when Netscape does someting, you need to be aware of it. They don't feature a lot of resources, but what is featured is typically pretty good.

Site: Dimension X
Location: http://www.dimensionx.com/dnx/java.html
Description: Dimension X is creating some of the coolest applet technologies on the Web. They have an awesome VRML environment that was written en-

Netscape - [New Media Marketing's CAFE DEL SOL]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://www.xm.com/cafe/
Welcome to the Café del Sol
The New Media Marketing lab's exhibition of Java Applets for the Web.
CREATORS
APPLETS
FAQ
RESOURCES
RULES
DOWNLOADING
Deep within the bowels of Sun Microsystems, Inc. lies a lab... the New Media Marketing lab. In addition to many other responsibilities the members of this lab are charged with the task of developing Java programs. These programs, innocently known as applets, are occasionally released... bringing much joy and happiness to the virtual world. Those with a keen eye can spot these applets on Web pages and in CDs released by the developers within New Media Marketing lab.
The Café del Sol is the first bit of the 'outside world' the applets see. Here they congregate, compile amongst themselves, and finally (with the permission of their creators) venture out amongst the other Kilo-Mega-Gigabytes that whiz by the Café del Sol, Sun Microsystems, and every other place on the Internet.
But here they reside first; here in their primal stages. If you are fortunate enough to have found the Café del Sol, then you are welcome to look around, investigate the applets, and basically get your fill of Java.
[CREATORS] - [APPLETS] - [FAQ] - [RESOURCES] - [RULES] - [DOWNLOADING]
All applets in the Café del Sol are © 1996, Sun Microsystems, Inc.
Café del Sol artwork, maintenance, and typos by
Eric Harshbarger -- harshec@cdware.eng.sun.com
sun microsystems
http://www.xm.com/cafe/resources.html

Figure A.5
Café Del Sol Java site.

Netscape - [Java Applets]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://home.mcom.com/comprod/products/navigator/version_2.0/java_applets/index.html
NETSCAPE NAVIGATOR
JAVA APPLETS
Netscape Navigator 2.0 supports Java on Sun Solaris, Sun OS, SGI IRIX, OSF/1, HP-UX, Windows NT, and Windows 95 platforms. Netscape anticipates shipping Java support for Windows 3.1, Macintosh, and AIX in the second quarter of 1996. A public beta release of Mac Java is due in late February. Using Java, Sun's new programming language based on C++ and designed for secure two-way, real-time interaction, developers can write custom mini-applications called Java applets. When integrated into Web pages, Java applets allow expert graphics rendering, real-time interaction with users, live information updating, and instant interaction with servers over the network. Java applets are downloadable from any server and run safely on any platform. Java is designed to provide maximum security on public networks, with multiple safeguards against viruses, tampering, and other threats.
Using Java, developers can write custom mini-applications called Java applets, which will provide Internet sites with a huge range of new functionality: animation, live updating, two-way interaction, and more. Java applets allow cross-platform programmability and can be embedded right into HTML pages.
Enter your Java applets in Sun's Java Cup International, cosponsored by Netscape.
JAVA APPLET DEMOS
To view these demos, be sure you've downloaded a client with Java support.
Blinking Text
Curve
Crossword
Graph Layout
Learn to Dance
Modern Clock
Simple 3D Viewer
Scrolling Images
StarField
Traditional Clock
Tic Tac Toe
Vector Graphic Manipulation
"Work in Progress" Animation
The Impressionist
JAVA RESOURCES
Document: Done

Figure A.6

Netscape's Java page.

tirely in Java called Liquid Reality. They also have several other applet creation utilities that do very specific, but very useful activities. For example, they have a Java program that helps create animator applets and another program that interfaces with the IRC protocol. By the time you read this they will probably have many more cool things! Figure A.7 shows a sample of the Dimension X Web site.

Site: Sun User's Group - Java Site
Location: http://www.sug.org/java-sig.html
Description: This Java site belongs to the official Sun User's Group (SUG). It is extremely comprehensive and has some really cool content as well. It is also the home of the Java Special Interests Group (Java SIG). Make sure you join their mailing list to get the latest information.

Site: The Coffee Pages
Location: http://answer.questions.com/date/960101/date.html
Description: Any description of this site as shown in Figure A.8 could not possibly do it justice so you need to go there and see it for yourself!

Netscape - [Dimension X Java(TM)]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://www.dimensionx.com/dnx/java.html
Dimension X Java(tm)
NOTE: This page and the examples below make use of the Java(tm) language developed at Sun Microsystems. Without a Java(tm)-capable browser the contents are a bit boring, as you will be unable to view the inline applications. For further information on Java(tm) see http://java.dimensionx.com/.
The following applets and packages run under the (frozen) "Beta API" specification for the Java(tm) language. They require a browser like Netscape 2.0 or the Appletviewer, part of Sun's JDK-1.0. They will not work with HotJava(tm) (which currently only exists for the Alpha API).
Liquid Reality
<lr@dimensionx.com>
Liquid Reality is package consisting of a browser for viewing and a toolkit for creating dynamic, interactive VRML worlds that can be extended with the Java(tm) language. Liquid Reality is based upon Ice, our 3D graphics package for use with Java(tm)-enabled browsers. There are two parts to Ice: a C library which implements low-level graphics primitives, and a set of Java(tm) classes which provide a portable interface to the C library. The current beta version is now available for download.
TEA (and JACK)
Patrick Schmitz <tea@dimensionx.com>
TEA (The Easy Animator) is an authoring tool for creating animations in Java(tm). It's a drag and drop tool that lets you create animated scenes with backrounds and soundtracks, sprites with different behaviors, drawn paths, etc. TEA uses JACK (the program formerly known as JAM/JAG) classes and is written completely in Java(tm), running on any platform the Java(tm) runtime has been ported to.
Applet ImageStripLoop running

Figure A.7
Dimension Liquid X.

Publications

Site: JavaWorld
Location: http://www.xm.com/cafe/
Description: *JavaWorld* is a new online magazine devoted to, guess what, Java! It features many good articles and links to more resources. The *zine* is set up similar to a traditional magazine with press releases, features, columns, reviews, and so on. This site is a good place to hang out to find out the latest news on the Java industry. Figure A.9 shows the latest issue.

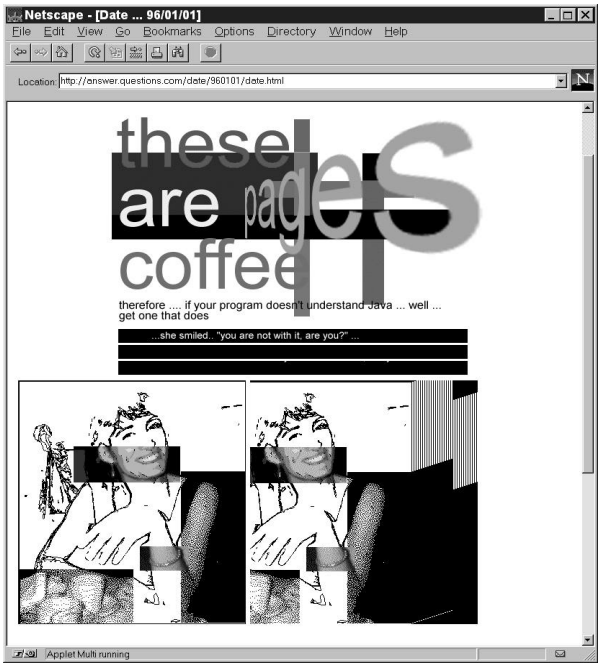

Netscape - [Date ... 96/01/01]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://answer.questions.com/date/960101/date.html
these pages are coffee
therefore .... if your program doesn't understand Java ... well ... get one that does
...she smiled.. "you are not with it, are you?" ...
Applet Multi running

Figure A.8
The Coffee Pages Web site.

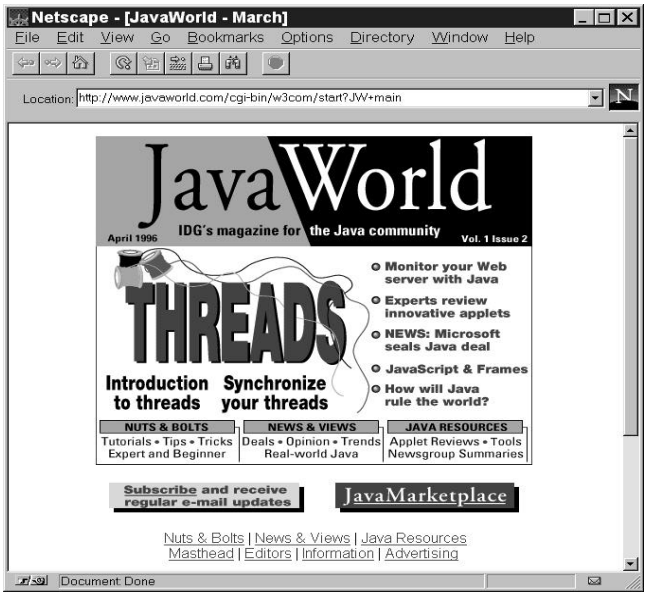
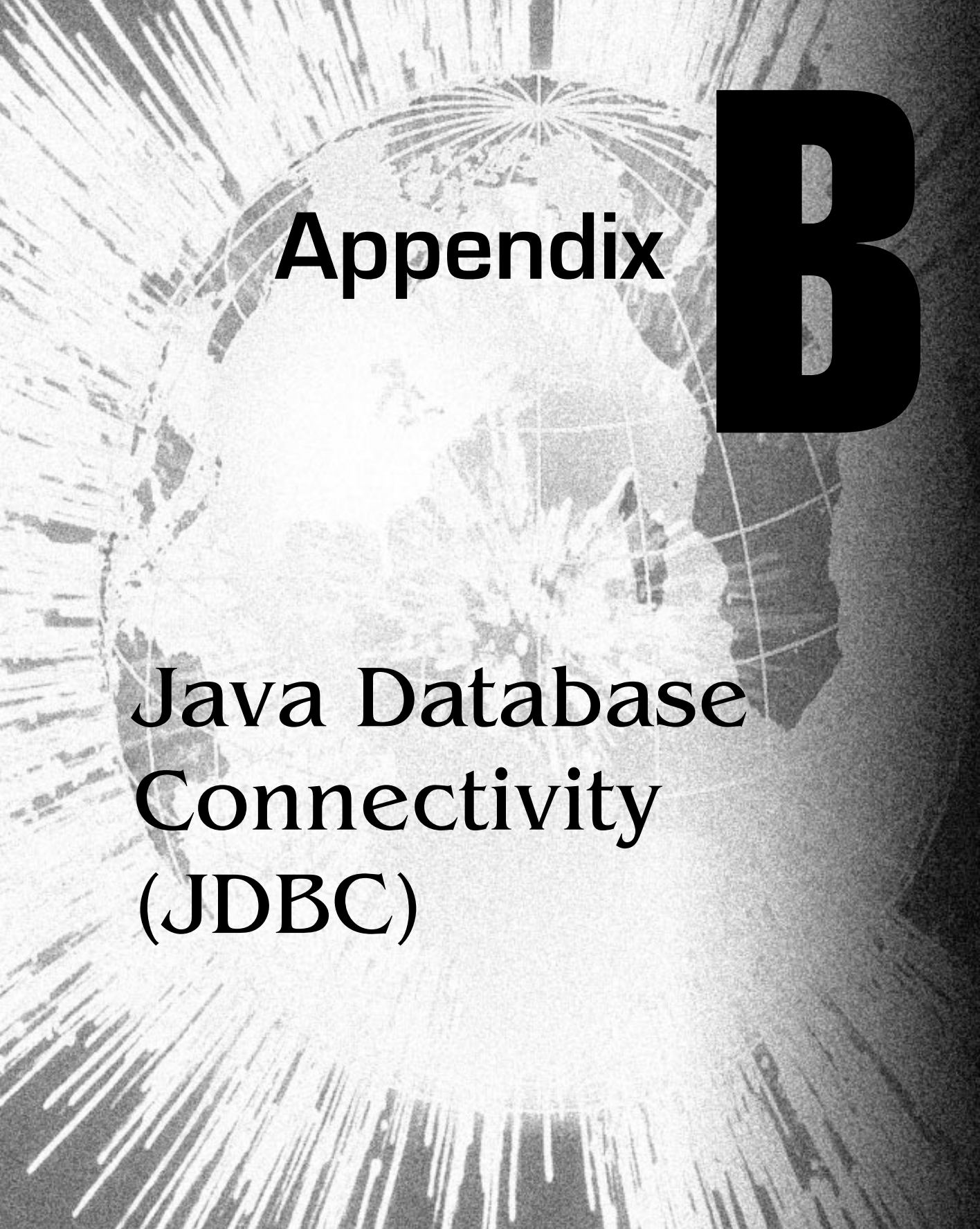

Netscape - [JavaWorld - March]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://www.javaworld.com/cgi-bin/w3com/start?JW+main
JavaWorld
April 1996
IDG's magazine for the Java community
Vol. 1 Issue 2
THREADS
Introduction to threads
Synchronize your threads
Monitor your Web server with Java
Experts review innovative applets
NEWS: Microsoft seals Java deal
JavaScript & Frames
How will Java rule the world?
NUTS & BOLTS
Tutorials • Tips • Tricks
Expert and Beginner
NEWS & VIEWS
Deals • Opinion • Trends
Real-world Java
JAVA RESOURCES
Applet Reviews • Tools
Newsgroup Summaries
Subscribe and receive regular e-mail updates
JavaMarketplace
Nuts & Bolts | News & Views | Java Resources
Masthead | Editors | Information | Advertising
Document Done

Figure A.9
The *JavaWorld* online magazine.



Appendix B

Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC)

B

As we were finishing the manuscript for this book, the developers of Java announced the release of the SQL API specifications. This next step in the evolution of Java may have a huge impact on how programmers use Java to build applications. The problem with Java so far is that it is not mature and robust enough to be used for developing large scale, real-world applications. For example, you probably wouldn't consider using Java to create a company-wide client/server system. But with the introduction of the Java Database Connectivity feature (JDBC), Java is on its way to offering everything developers need to implement database schemes that are fast enough for everyday use.

By implementing a database standard for Java developers to follow, Sun has really helped Java's chances of becoming a language that will be used extensively in the future. To maintain as much of its cross-platform capabilities as possible, Sun is pushing a generic SQL database access framework. This will provide a uniform interface that can run on top of different database systems. This means that the tools you use in Java should always look the same, but the drivers that actually connect to the databases can be anything you want. As new database standards emerge, the JDBC will be able support them as soon as the appropriate low-level drivers are written.

Of course, keep in mind that the specifications for the new JDBC standard have just begun and they are likely to evolve in the near future. This first draft of the specifications is designed to help developers create database drivers more than it is designed to help database users.

In this appendix, we'll introduce you to this new technology so that you can get a sense of how this powerful database API is evolving. You can think of it as sneak preview of what is to come. At this stage, we don't recommend that you create any huge database applets or applications based on this API. You should first experiment with it and become familiar with it so that when the 1.0 version of the API is released you will be ready to jump right into it with both feet.

Introducing the Database API

At the time of this writing, Version 0.5 of the API was introduced. By the time your read this chapter, a newer version may be available. Make sure you check out the Javasoft Web site (www.javasoft.com) from time to time to look for new developments. If you are inexperienced with SQL database technology, we suggest you pick up a good SQL tutorial book. We couldn't possibly cover the entire SQL system in one chapter, so we will assume that you have at least a basic understanding of SQL.

The Java Database Connectivity (JDBC) API is based on four simple ideas. First, it offers an *environment* that provides a support structure for connecting to a database (or multiple databases). Next, it provides a *connection* that initiates contact and communication between a client and a server. Third, it accepts *statements* for specifying parameters for performing queries and searching databases. Finally, it returns a *result set* that stores the results of a query performed by a statement.

Security Issues

One of the strong suits of Java is its robust security measures. The JDBC API should be just as secure. If you use a database from within an applet, it will have the same constraints as file handling: *You will only be able to connect to a database that is on the server from which the applet was called from.* However, security "certificates" may be used to allow trusted applets to have more access.

One area of security that may be problematic involves using a database that can store binary large objects (BLOBs). These structures can store anything including images, video, animation, and unfortunately viruses. It is feasible that a misguided hacker could use a database system to trans-

fer a virus or other harmful bit of native code onto any system through the use of a BLOB. This possibility will undoubtedly be looked at by the developers of the JDBC API but you should realize that potential dangers exist for using distributed database systems in general. As you start to make use of this technology in building your own applications, keep your eyes open for potential security problems.

The Database Classes

Let's look at the classes and methods that make up the current JDBC API. The key classes included:

- Environment
- Connection
- Driver
- ResultSet
- Statement

The Environment Class

The **Environment** class is used as a backdrop for all the other operations we wish to perform within our database. We also use this class to determine which type of driver/database combination we will be using. At this time, only the standard JDBC driver is available, but there will soon be drivers for many different database connectivity systems. Here is the declaration for this class:

```
public Environment();
Environment myEnv = New Environment();  // sample declaration statement
```

The methods for the **Environment** class are listed in Table B.1.

The Connection Interface

The **connection** interface is used to create driver-specific classes that can be used to interact with each different driver in exactly the same way. The **connection** interface and any classes that implement this interface will have the static vari-

Table B.1 The Methods for the Environment Class

Method	Description
getConnection(String, Properties)	This method initiates the connection to the database that is in the form of a URL passed through the **String** argument. It will try and make this connection by cycling through each registered database driver. The **Properties** argument contains user information such as an ID and password.
getConnection(String, String, String)	This method is identical to the one above, except that the **Properties** argument has been split up into its parts; strings representing the user ID and password.
getDrivers()	This method returns an enumeration of all the currently available JDBC drivers. Calling this method is a good way to check if a particular driver is loaded and if not you could try and download it by calling the **loadDriver()** method.
loadDriver(String)	This method tries to load a new driver. The string passed specifies the full classname (including the .class) of the driver we want to try and load. The method returns a boolean value indicating the success or failure of the load.
registerDriver(Driver)	This method registers new drivers when they are downloaded.

Table B.2 Variables Used with the Connection Interface

Variable	Value
TRANSACTION_NONE	0
TRANSACTION_READ_UNCOMMITTED	1
TRANSACTION_READ_COMMITTED	2
TRANSACTION_REPEATABLE_READ	3
TRANSACTION_SERIALIZABLE	4
TRANSACTION_VERSIONING	5

ables and methods listed in Tables B.2 and B.3. Keep in mind that the variables are all static variables belonging to the interface. They are used for setting and checking the status of database read methods.

The Driver Interface

The **driver** interface is used by database driver developers. You will not need to use it unless you are creating your own drivers. However, all JDBC drivers will need to implement the methods listed in Table B.4.

Table B.3 Methods Used with the Connection Interface

Method	Description
close()	This method frees up resources linked to the current connection. These resources can be garbage collected eventually, but it is better to do it manually to avoid memory problems.
commit()	This method commits the stored SQL statements to be queried.
createStatement()	This method simply returns a database SQL object.
getTransactionIsolation()	This method returns an integer value representing the isolation level of the connection.
isClosed ()	This method returns a boolean value that indicates if the database is closeed or not. A closed database can occur due to a call to the **close()** method or through an error with the connection.
isReadOnly()	This method checks the read-only status of the connection. The return value is a boolean.
nativeSQL(String)	This method takes a standard SQL query and converts it into a query in the form of a string that is ready for the driver specific to this class. This method is necessary if you are using a database driver that uses proprietary SQL calls.
prepareCall(String)	This method is very similar to **prepareStatement()** but it returns a **CallableStatement** object which is a call to a stored SQL procedure. Once again, the **String** argument is simply the SQL query you wish to send.
prepareStatement(String)	This method returns a **PreparedStatement** object which is a pre-compiled SQL statement. This type of statement increases query speed in large databases. The **String** argument is simply the SQL query you wish to send.
rollback()	This method will negate all the queries that are stored in the queue.
setAutoCommit(Boolean)	This method tells the database driver that all SQL statements will automatically be executed as they are prepared. Otherwise, each statement is put into a list that is not sent until the **commit()** method is called.
setReadOnly(boolean)	This method sets the read-only attribute for the connection. This method cannot be called during the middle of a transaction.
setTransactionIsolation(int)	This method sets the transaction isolation level. The integer being sent can be represented by one of the interface variables listed above. Once again, this method cannot be called in the middle of a transaction.

Table B.4 The Methods Used with the Driver Interface

Method	Description
connect(String, Properties)	This method returns a **Connection** object that can then be used to create queries. The **String** argument is the URL of the desired database and the **Properties** argument is used to pass arbitrary information, usually username and password.
getMajorVersion()	This method returns an integer that represents the major version number of the driver.
getMinorVersion()	This method returns an integer that represents the minor version number of the driver.

The ResultSet Interface

The objects you will use to actually interact with database information will be implementations of the **ResultSet** interface. From this level, you will be able to read an incredible number of different types of data. Look near the end of the chapter for information on the SQL data types. The methods defined in this interface are listed in Table B.5.

Table B.5 The Methods Used with the ResultSet Interface

close()	This method closes the result set so that its resources can be recycled.
getBigInt(int)	This method accepts a column number as an argument and returns a **long** value.
getBinary(int)	This method accepts a column number as an argument and returns an array of **byte** values.
getBit(int)	This method accepts a column number as an argument and returns a **boolean** value.
getChar(int)	This method accepts a column number as an argument and returns a **String** object.
getColumnCount()	This method returns an integer representing the number of columns in the database.
getDate(int)	This method accepts a column number as an argument and returns a**Date** value. (**Date**, **Time**, and **Timestamp** are three more new data classes introduced with the SQL package.)
getDecimal(int, int)	This method accepts a column number and scale value as arguments and returns a **Numeric** value.
getDouble(int)	This method accepts a column number as an argument and returns a **double** value.
getFloat(int)	This method accepts a column number as an argument and returns a **float** value.
getInteger(int)	This method accepts a column number as an argument and returns an **integer** value.
getLongVarChar(int)	This method accepts a column number as an argument and returns a **String** object.
getNumeric(int, int)	This get method is a little different. First, we have two arguments. The first is the column number and the second is the scale of the value—the number of decimal point to the right to use for accuracy. The return value for this method is a little different also. Instead of getting a standard data type, you get a new object, **Numeric**. **Numeric** objects can be used to retrieve various types of numeric values and parse out the type later. This method is great for dealing with databases where you are not sure of some column data types.
getReal(int)	This method accepts a column number as an argument and returns a **float** value.
getRowCount()	This method counts the number of total rows in this result set.
getRowNumber()	This method returns an integer representing the current row number in the result set.
getSmallInt(int)	This method accepts a column number as an argument and returns a **short** value.
getTime(int)	This method accepts a column number as an argument and returns a **Time** value.
getTimestamp(int)	This method accepts a column number as an argument and returns a **Timestamp** value.

continued

Table B.5 The Methods Used with the ResultSet Interface (Continued)

getTinyInt(int)	This method accepts a column number as an argument and returns a **byte** value.
getVarChar(int)	This method accepts a column number as an argument and returns a **String** object.
isNull(int)	This method returns a boolean value that tells us if the column at the given **integer** is null or not.
next()	This method advances to the next record in a result set. It returns true if the next record is available or false if the end of result set is reached.

The Statement Interface

The **Statement** interface is used for executing queries that will result in a**resultSet**. The methods defined in this interface are listed in Table B.6.

Table B.6 The Methods Used with the Statement Interface

Method	Description
cancel()	This method can be used in a threaded program to stop the execution of a query that is being processed by another thread.
clearWarnings()	This method clears the warnings from the queue.
close()	This method closes all resources associated with a **statement** object.
execute(String)	This is method is the counterpart to **executeQuery()**. It is used to perform statements on the database that write to it rather than read. The return value is an integer representing the new total number of rows for the database.
executeQuery(String)	You will probably use this method more than any other for retrieving information from a database. Its sole argument is a **String** that holds a standard SQL query. After this method executes successfully, a **resultSet** object is returned.
getMaxFieldSize()	This method returns an integer value that indicates the limit in bytes on how much data can be sent as part of any one single field.
getMaxRows()	This method returns an integer value that indicates the maximum number of rows that can be returned as part of a **resultSet**.
getWarnings()	This method returns error information relating to the current statement either being executed or has executed. This can be a single or multiple errors of the **SQLWarning** type.
setMaxFieldSize(int)	This method sets the maximum number of bytes that can be transferred with any single field.
setMaxRows(int)	This method is used to set the maximum number of rows that can be received as part of a **resultSet** from a query.

The Numeric Class

The **Numeric** class can be used to represent SQL values from a database before they are transferred to a Java data type. This class is useful for performing operations on SQL specific data that will be returned to the database and will not be needed in a Java-centric data type.

This class includes its own simple math functions as well. However, whenever you perform a mathematical operation with one or more **Numeric** objects, you will always receive another **Numeric** object as the return type. Table B.7 describes the methods found in this class.

The **Numeric** object is quite a bit different than using our standard data types, so let's look at a few simple code snippets that illustrate the use of the **Numeric** class:

```
Numeric x, y, z, result;
boolean bool;

x = new Numeric(5, 0);              // x = 5 (integer)
y = new Numeric(5.0, 1);            // y = 5.0 (double)
z = new Numeric(2.5, 4);            // z = 2.5000 (double)
result = x.add(y);                  // result = 10
result = y.subtract(z);             // result = 2.5
result = z.multiply(z);             // result = 6.2500
bool = y.greaterThan(z);            // bool = true
bool = x.lessThanOrEquals(z);       // bool = false
```

The Date Class

One type of data element that Java does not support very well are dates and times. Since these are improtant parts of the SQl database system, Sun created three new data type classes. The constructors and methods for these classes are presented in Table B.8.

The Timestamp Class

The **Timestamp** data type is the **Time** and **Date** data types mixed together and put on steroids. You can use it to represent times down into the nanosecond range as well as the date. The constructor for this class is:

```
Timestamp(int, int, int, int, int, int, int);
```

Table B.7 The Methods Used with the Numeric Class

Method	Description
add(Numeric)	Adds the given **Numeric** value to the current one. The result is a third **Numeric** object.
divide(Numeric)	Divides the given value into the current value.
Environment(double, int)	Creates a double style object.
Environment(int, int)	Creates an integer-based representation of the database value.
Environment(String, int)	Creates a string representation of the value with the integer being the scale of the numeric accuracy.
equals(Object)	Returns true if the numeric value matches the objects value; otherwise, it returns false.
greaterThan(Numeric)	Returns a boolean value.
greaterThanOrEquals(Numeric)	Returns a boolean value.
lessThan(Numeric)	Returns a boolean value.
lessThanOrEquals(Numeric)	Returns a boolean value.
multiply(Numeric)	Performs a multiplication operation.
subtract(Numeric)	Same as **add()** but performs subtraction.

Table B.8 The Constructors and Methods Used with the Date and Time Classes

Method	Description
Date(int, int, int)	The constructor for the **Date** class. The three integer arguments represent the year, month, and day, respectively.
toString()	This is the only method available to the **Date** class. It returns a string representation of the date.
Time(int, int, int)	The constructor for the **Time** class. The three integer arguments represent hours, minutes, and seconds, respectively. The **Time** class uses 24-hour military time.
toString()	This is the only method available to the **Time** class. It returns a string representation of the time.

The list of integer arguments include the year, month, day, hour, minute, second, and nanoseconds. The only method provided in this class is **toString**(), which returns a string representation of the date and time stored in the object.

SQL Data Types

When accessing SQL databases your programs will come into contact with data types that do not exist as standard data types in Java. For this reason, the JDBC will automatically *map* the SQL data type to the closest Java data type. If you want to handle this yourself, you can actually cause data types to be mapped into many possible Java types. Table B.9 presents the set of SQL data types used along with their corresponding Java data types.

After examining Table B.9, you're probably wondering about the **InputStream** data types that are listed. They are an object that can be used to stream data into a program as needed. Sometimes a database will have fields that store huge amounts of data like blobs that if we tried to download them into a **resultSet** we

Table B.9 The Set of SQL Data Types Along with Their Corresponding Java Types

SQL Data Type	Java Object Type
Char	String
VARCHAR	String
LONGVARCHAR	InputStream
NUMERIC	Numeric
DECIMAL	Numeric
BIT	boolean
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Long
REAL	Float
FLOAT	Float
DOUBLE	Double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	InputStream
DATE	Date
TIME	Time
TIMESTAMP	Timestamp

would run out of memory and have to wait a very long time for the data to come across the Net. The **InputStream** class works by reading the data in manageable chunks that the system can handle.

JDBC URL Usage

A few of the classes and methods we described in the previous section use **String** objects that represent URLs. These URLs are quite different than the standard URLs used for Web and FTP navigation. This part of the JDBC API is not set in stone yet so watch out for big changes here.

As it stands now, here is the syntax for a JDBC URL:

```
jdbc:<subprotocol>:<subname>
```

The *subprotocol* should name the type of database protocol you are running SQL through. The *subname* will represent more detailed information that the subprotocol can use to locate a specific server or database.

The *subname* can further be refined to include information about the location of a secondary server on a LAN. Here is an example of this:

```
jdbc:odbc://ntserver:80/web
```

If this looks confusing, don't panic yet. Wait for the full release and hopefully some more detailed documentation from Sun. Also, we expect that there will be a tremendous number of tools available to aid in this sort of development. For example, the development environments that Symantec and Borland are working on will undoubtedly aid in database application and applet development.

Eventually, we may have protocol naming services, much like the dynamic naming services we have now for Web sites. So, you may be able to use a simple name that points to your server over the Web by resolving the name with a naming service. At this point, Sun is going to start a registry that people can register their drivers with.

JDBC Code Examples

Here are a couple of simple classes that illustrate how the database system will work. Note that because we don't have easy access to a SQL server that you and we can connect to, the URLs in these samples are provided for example purposes only.

Retrieving Information from a Database

This first example shows you how to use the SQL package to create database queries for retrieving information. In particular, we use the **Connection**, **Statement**, and **ResultSet** classes.

```
import java.net.URL;
import java.sql.*;

class readDatabase {

  public static void main(String args[]) {
    try {
      // Create a URL specifying an ODBC database
      String url = "jdbc:odbc:coriolis";
      // Connect to the database
      Connection con = Environment.getConnection(url, "username", "password");
      // Create and execute a SQL SELECT statement
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT a, b, c, d, key FROM
        Table");
      // Step through the result
      while (rs.next()) {
        // get the values from the current row:
        int a = rs.getInteger(1);
        Numeric b = rs.getNumeric(2);
        char c[] = rs.getVarChar(3).tocharArray();
        boolean d = rs.getBit(4);
        String key = rs.getVarChar(5);
        // Now print out the results:
        System.out.print(" key=" + key);
        System.out.print(" a=" + a);
        System.out.print(" b=" + b);
        System.out.print(" c=");
        for (int i = 0; i < c.length; i++) {
          System.out.print(c[i]);
        }
        System.out.print(" d=" + d);
        System.out.print("\n");
      }
      stmt.close();
      con.close();
    } catch (java.lang.Exception ex) { ex.printStackTrace(); }
  }
}
```

Writing to a Database

In this example we are using classes from the SQL package to write data to a database:

```
import java.net.*;
import java.sql.*;

class writeDatabase {
  public static void main(String args[]) {
    try {
      // Create a URL specifying an ODBC database
      String url = "jdbc:odbc:coriolis";
      // Connect to the database at that URL.
      Connection con = Environment.getConnection(url, "username", "password");
      // Create a prepared statement to update the third field in a
      // row in the "Table" table.
      PreparedStatement stmt = con.prepareStatement( "UPDATE Table1 _
        SET a = ? WHERE key = ?");
      // First use the prepared statement to update the "count" row.
      stmt.setInteger(3, 34);
      stmt.setVarChar(2, "count");
      stmt.execute(); //Executes the query and update
      stmt.close();
      con.close();
    } catch (java.lang.Exception ex) { ex.printStackTrace(); }
  }
}
```

Index

A

Abstract classes, 33, 121, 158
Abstract methods, 132
Abstract Window Toolkit, 32
Action( ) method, 294
Add method, 274
Addition, 98
Addressing
  Internet, 353
Animation
  buffering, 40
  speed, 26
API documentation, 64
Applet class
  hierarchy, 288
  methods available, 288
  methods derived, 290
Applets, 5, 6
  browser interaction, 294
  class, 287
  closing, 51
  compiling, 53
  defined, 21
  drawbacks, 291
  file access, 292
  file execution, 292
  fonts, 43
  images, 298
  navigation sample, 293
  network communication, 292
  package, 31
  parameters, 39
  passing information, 367
  sounds, 297
  tags, 39
  threading, 51
  ticker tape sample, 22
  vs. applications, 21
Appletviewer, 7
Applications
  command line arguments, 86
  defined, 21
  networking, 353
  sample menu, 268
  vs. applets, 21
Architecture natural, 5
Arguments
  accessing, 88
  indexing, 89
  numeric, 89
  passing, 87
  reading, 87
Arrays, 16, 82
  accessing data, 86
  declaring, 82
  elements, 83
  indexing, 85
  multidimensional, 85
  sizing, 83
Assignment boolean operators, 102
Assignment operators, 93, 95
Audio clips, 297
AWT, 31, 227
  AWTError, 191

class hierarchy, 230
components, 229
defined, 32
importing, 228
layout manager, 228
menus, 229

B

Bandwidth considerations, 360
Binary, 97
Binary integer operators, 99
Binary operators, 98
Bitwise complement, 97
Bitwise operators, 99
Blocking, 217
Body (class), 128
Boolean data type, 78
Boolean operators
assignment, 102
evaluation, 101
logical, 100
negation, 100
ternary, 102
BorderLayout class, 274
declaration, 275
methods, 275
Break statement, 110
Browsers
HotJava, 6
Netscape, 26
BufferedInput Stream class, 327
BufferedOutputStream class, 327
Buffering, 40, 45
Button class
declaration, 243
getLabel( ), 244
hierarchy, 243
setLabel( ), 244
Buttons, 32
Byte streams, 321
Byte type, 76
ByteArrayInputStream class, 328
ByteArrayOutputStream class, 328
Bytecodes, 5, 53

C

Canvas class
declaration, 245
hierarchy, 244
paint( ), 245
CardLayout class, 282
declaration, 282
methods, 282
Case-sensitivity
declarations, 36
package names, 171
parameters, 40
variable names, 36
Casting
interfaces, 165
vs. creating, 151
Casts, 103
Catch statements, 187
Catch( ) method, 185
Catching errors, 186
CGI. *See Common Gateway Interface*
Char data type, 79
Character arrays, 16
Character literals, 73
Checkbox class, 245
declaration, 246
getCheckboxGroup( ), 247
getLabel( ), 247
getState( ), 247
hierarchy, 246
setCheckboxGroup( ), 247
setLabel( ), 247
setState( ), 247
Choice class, 247
addItem( ), 249
countItems( ), 249
declaration, 248, 251
getItem( ), 249

getSelectedIndex( ), 249
getSelectedItem( ), 249
hierarchy, 248, 250
methods, 251
select( ), 250
Classes, 5
abstract, 33, 121, 158
advantages, 116
applet, 287
body, 128
bufferedInputStream, 328
bufferedOutputStream, 327
button, 243
byteArrayInputStream, 328
byteArrayOutputStream, 328
canvas, 244
casting, 150
checkbox, 245
choice, 247
component, 290
container, 290
dataInputStream, 330
dataOutputStream, 330
declaring, 116
defined, 32
documenting, 63
error, 191
event, 304
exception, 193
extending, 124
fileInputStream, 333
fileOutputStream, 333
filterInputStream, 335
filterOutputStream, 335
final, 33, 123
flowLayout, 271
frame, 235
fully qualified name, 118
hiding, 177
identifiers, 124
importing packages, 176
InetAddress, 354
inputStream, 325
label, 241
lineNumberInputStream, 337
list, 250
menuItem, 265
modifiers, 33, 119
name space, 34, 129
naming, 124
networking, 353
object, 34
outputStream, 325
panel, 238
pipedInputStream, 339
pipedOutputStream, 339
printStream, 340
private, 33
protocols, 158
public, 33, 120
pushbackInputStream, 342
runtime, 194
scrollbar, 258
sequenceInputStream, 342
socket, 355
stringBufferInputStream, 343
super( ), 142
superclass, 34
System, 321
textArea, 253
textField, 253
throwable, 182
URL, 364
variables, 148
WriteAFile, 185
CLASSPATH, 171, 173, 174
Client, 350
Client/server technology, 350
Code parameter, 27
Color method, 37
Command line arguments, 86
indexing, 89
numeric, 89
passing arguments, 87
reading, 87
Comments, 30
styles, 59
tags, 67

Common Gateway Interface, 10
Compilers, 7, 53
Component class
  bounds( ), 232
  disable( ), 232
  enable([Boolean]), 232
  getFontMetrics( ), 232
  getGraphics( ), 232
  getParent, 232
  handleEvent(Event evt), 232
  hide( ), 233
  inside(int x, int y), 233
  isEnabled( ), 233
  isShowing( ), 233
  isVisible( ), 233
  locate(int x, int y), 233
  location( ), 233
  move(int x, int y), 233
  repaint( ), 233
  resize( ), 234
  setFont( ), 234
  show([Boolean]), 234
  size( ), 235
Components, 60
Compound expressions, 104
Compound statements, 106
Constructors, 37, 138
  body, 146
  calling, 140
  declaring, 140
  FontMetrics, 48
  Java default, 142
  modifiers, 143
  object creation, 148
  protected, 143
  public, 143
  return type, 139
Container class, 290
Control flow, 106
Control structures
  do...while, 108
  for, 110
  if...else, 106
  labels, 111
  list of, 107
  switch, 109
  while, 108
Controls, 229
  buttons, 243
  canvas, 244
  checkbox, 245
  components, 231
  frame, 235
  label, 241
  layout manager, 270
  lists, 250
  menus, 229, 263
  panel, 238
  pop-up menus, 247
  scrollbar, 258
  text areas, 253
  text fields, 253
Converting values
  casting, 150

D

Data types, 35
  boolean, 78
  byte, 76
  casting, 103
  char, 79
  double, 78
  float, 78
  int, 71, 77
  long, 71, 77
  separators, 75
  short, 76
  string, 79
  variables, 76
DataInputStream class, 330
DataOutputStream class, 330
Debugging, 181
Decrement operator, 98
Destroy( ) method, 222
Developers Kit, 17
Directories

search path, 174
Disassembler program, 17
Distributed programming, 6
Distributed software, 10
Division, 98
Do...while, 108
Doc comment clauses, 119
Documenting classes, 63
Double buffering, 45
Double data type, 78

E

Encapsulation, 43
Equal to operators, 102
Error handling, 181
Errors
  catching, 186
  checking, 323
  file not found, 189
  input/output, 185
  throws, 133
  try clauses, 186
Evaluation operators, 101
Event class, 304
  methods, 306
  variables, 305
Event handling, 53
Events
  hierarchy, 313
  processing problems, 318
  system, 315
  types, 304
Exceptions, 15, 181, 182
  class, 193
  creating, 200
  error, 191
  file not found, 189
  finally statements, 189
  handler, 182
  IOException, 185
  try clauses, 186
  URL, 364
Executable content, 10
Export statement, 228
Expressions
  assignment, 105
  writing, 104
Extending classes, 124
Extends keyword, 34

F

Fatal errors, 191
File
  access, 292
  execution, 292
  input/output, 321
  saving, 173
File Transfer Protocol. *See FTP*
FileInputStream class, 333
FileOutputStream class, 333
FilterInputStream, 335
FilterOutputStream class, 335
Final classes, 33
Final methods, 132
Finally statement, 189
Finger protocol, 349
Float data type, 78
Floating-point, 72
  operators, 102
FlowLayout class, 271
  declaration, 271
  methods, 273
Font metrics, 48
Fonts, 43
For loops, 110
Frame class, 235
  declaration, 235
  dispose( ), 237
  getIconImage( ), 237
  getMenuBar, 237
  getTitle( ), 237
  hierarchy, 235
  isResizable( ), 238
  remove( ), 238

setCursor( ), 238
setIconImage( ), 238
setMenuBar( ), 238
setResizeable( ), 238
setTitle( ), 238
FTP, 349

G

Garbage collection, 6, 15, 37
Gateways, 355
Graphical User Interface
button class, 243
canvas class, 244
checkbox class, 245
choice class, 247
component class, 231
frame class, 235
label class, 241
lists, 250
menu class, 263
menu items, 265
menuBar class, 261
panel class, 238
scrollbar class, 258
text areas, 253
text fields, 253
Graphics methods, 46
GridBagLayout class, 278
declaration, 281
methods, 281
variables to customize, 278
GridLayout class, 276
declaration, 277
methods, 277

H

Header files, 13
Height parameter, 27
Helper programs, 17
Hexadecimal format, 71
History of Java, 8
HotJava, 6, 10
HTML. *See Hyper Text Markup Language*
applet tags, 39
HTTP, 349
Hyper Text Markup Language, 25
Hyper Text Transfer Protocol. *See HTTP*

I

Identifiers, 65
class, 118
classes, 124
errors, 67
If...else, 106
Image buffer, 41
Images, 298
Implements clause, 126
Implements keywords, 34
Import statements, 31, 228
Including packages, 31
Increment operator, 98
Index, 84
InetAddress class, 354
Init( ), 130
Input streams, 321, 324
InputStream class, 325
methods, 325
Instanceof operator, 17, 168
Int data type, 77
Integers, 71
literals, 72
operators, 93, 97
Interfaces, 34, 158
casting, 165
class, 126
declaring, 161
design issues, 160
implementation tips, 167
implementing, 161
implements clauses, 126
keyword, 161

layout manager, 271
runnable, 34
tips on using, 165
Internet
addressing, 353
java.net package, 352
Request for Comments, 351
IOException, 324

J

Java language
advantages, 4
benefits, 11
compared to C++, 9
developer's kit, 7, 17
history, 8
interfaces, 158
jargon, 5
tools, 8
virtual machine, 6
JAVAC, 7, 53
JAVADOC.EXE, 63
Java-enabled, 7
JAVAP, 17
JavaScript, 7
Just-in-Time compiler, 7

K

Keyboard events, 311
keyDown( ), 311
keyUp( ), 311
Keywords, 68
class, 124
extends, 34, 124
implements, 34, 162
interface, 161
list of keywords, 69
super, 135
this, 50, 135

L

Label class, 241
declaration, 241
getAlignment( ), 242
getText( ), 242
hierarchy, 241
setAlignment( ), 242
setText( ), 243
Labels, 111
Layout manager, 228, 270
borderLayout class, 274
cardLayout class, 282
flowLayout class, 271
gridBagLayout class, 278
gridLayout class, 276
Lexical structures, 58
comments, 59
identifiers, 65
keywords, 68
separators, 75
LineNumberInputStream class, 337
List class, 250
Literals, 71
character, 73
numeric, 71
Logical operators, 100
Long data type, 77
Long integers, 71

M

Main programs, 27
Menu class, 263
declaration, 264
hierarchy, 263
methods, 264
MenuBar class, 262
declaration, 262
hierarchy, 262
methods, 262
MenuItem class, 265

declaration, 266
hierarchy, 266
methods, 267
Menus, 32
creating, 229
Methods, 7, 38, 130
abstract, 132
action( ), 294
add( ), 274
applet class, 288
body, 134
catch( ), 185
color, 37
constructors, 138
createImage( ), 230
declaring, 130
defined, 28
destroy( ), 222
disable( ), 230
documenting, 63
drawString( ), 48
final, 132
getGraphics( ), 41
getMessage, 198
getParameter( ), 40
graphics, 46
handleEvent( ), 53
hide( ), 230
init( ), 28, 130
main( ), 87
modifiers, 131
native, 132, 292
overloading, 137
overriding, 43, 137, 170
paint( ), 29, 44
parameter lists, 133
parse( ), 89
private, 132
protected, 131
public, 131
resume( ), 220
return type, 133
Run( ), 29, 214
sleep( ), 50
start( ), 29
static, 132
stop( ), 51, 221
suspend( ), 220
synchronized, 132
throwing an exception, 194
throws, 133
valueOf( ), 89
write( ), 184
yield, 221
Modifiers
abstract, 121
constructor, 143
final, 123, 150
method, 131
modifiers, 33, 119
public, 120
transient, 150
volatile, 150
Modulus operator, 99
Mouse events, 304, 307
mouseDown( ), 307
mouseDrag( ), 309
mouseEnter( ), 310
mouseExit( ), 310
mouseMove( ), 309
mouseUp( ), 308
Multidimensional arrays, 85
Multiple inheritance, 14
Multiplication, 98
Multithreading, 7, 208
grouping, 226
synchronizing, 222

N

Name space, 129
Native methods, 132, 292
Negation operator, 97
Netscape
applet, 294
Network communication, 292

Network News Transfer Protocol.
See NNTP
Networking, 347
between applets, 367
classes, 353
client/server, 350
concerns, 360
java.net, 352
ports, 350
protocols, 348
sockets, 355
URLs, 364
New lines, 356
NNTP, 349
Not equal to operators, 102
Numeric literals, 71

O

Object-oriented programming, 12
Objects
arrays, 82
class, 34
creation, 148
declaring, 118
Octal integers, 71
Operators, 74, 93
addition, 98
assignment, 95, 102
binary, 98
binary integer, 99
bitwise, 99
bitwise complement, 97
boolean negation, 100
compound expressions, 104
decrement, 98
division, 98
equal to, 102
evaluation, 101
floating-point, 102
increment, 98
instanceof, 17, 168
integer, 97
logical AND, 100
modulus, 99
multiplication, 98
negation, 97
not equal, 102
precedence, 93
subtraction, 98
ternary, 102
Output streams, 324
class, 325
Overloading methods, 137
Overriding methods, 137

P

Packages, 30
applet, 31
awt, 31
case sensitivity, 171
classes, 30
creating, 168
documenting, 63
import keyword, 169
java.io, 322
java.lang, 30
java.net, 352
naming, 170
public classes, 172
standard Java, 177
Paint( ) method, 44
Panel class, 238
declaration, 240
hierarchy, 240
setlayout( ), 241
Parameter lists
constructor, 146
Parameters, 39
code, 27
height, 27
speed, 26
values, 27
width, 27
Parsing, 89

Performance issues
  threading, 51
PipedInputStream class, 339
PipedOutputStream class, 339
Pointers, 13
Ports, 350
  Internet, 351
  numbers, 350
Precedence (operators), 93
PrintStream class, 340
Private
  constructors, 143
  methods, 132
Processing parameters, 39
Protected
  constructors, 143
  methods, 131
Protocols
  class, 158
  finger, 349
  FTP, 349
  Internet, 351
  NNTP, 349
  Request for Comments, 351
  SMTP, 348
  TCP/IP, 348
  WhoIs, 349
Public
  classes, 33, 120
  constructors, 143
  keyword, 162
  method, 131
PushbackInputStream class, 342

R

Request for Comments. *See Request for Comments*
Resizing, 239
Resource allocation, 37
Resume( ) method, 220
Return type, 133
Returns, 356
RFCs. *See Request for Comments*
Run method, 214
Runnable interface, 213
Runtime class, 194

S

Savings files, 173
Scripting language, 7
Scrollbar class, 258
  hierarchy, 260
  methods, 260
Security, 12, 15, 292
Seprators, 75
SequenceInputStream class, 342
Servers, 350
  sample, 361
  setting up, 360
ServerSocket class, 360
Shadowing, 129
Short type, 76
Simple Mail Transfer Protocol. *See SMTP*
Simple statements, 105
Single inheritance, 121
Sleep( ) method, 50, 219
Socket class, 360
Sockets, 355
Sounds, 297
Source code
  saving, 173
Statements, 105
  catch, 187
  compound, 106
  control flow, 106
  finally, 189
  simple, 105
  switch, 109
  using semi-colons, 106
  writing, 104
Static methods, 132
Status bar, 296
Stop( ) method, 221

Streams, 321
  inputStream, 324
  outputStream, 324
String arrays, 16
String type, 79
StringBufferInputStream class, 343
Subclasses, 44
Subtraction, 98
Super classes, 16
Super keyword, 135
Super( ), 142
Suspend( ) method
  suspending execution, 220
Switch, 109
Synchronized methods, 132
System class, 321
  system.in, 322
System events, 315
  action( ), 317
  handleEvent( ), 317

T

Tags, 67
TCP/IP, 348
Ternary operators, 102
TextArea class, 253
  declaration, 254
  hierarchy, 254
  methods, 255
TextField class, 253
  declaration, 254
  hierarchy, 254
  methods, 255
This keyword, 50, 135
ThreadGroup, 226
Threads, 29, 49, 182, 207, 212
  blocking, 217
  creating, 211
  destroy( ) method, 222
  first in first out, 217
  grouping, 226
  initializing, 215
  life cycle, 218
  priority, 217
  resuming, 220
  run( ) method, 219
  runnable interface, 213
  sleep( ) method, 219
  start( ) method, 219
  stop( ) method, 221
  subclassing, 212
  suspending execution, 220
  synchronizing, 222
  when to use, 210
  while loops, 210
  yield( ) method, 221
Throws, 133
  constructor, 146
  exceptions, 194
Transient modifiers, 150
Transmission Control Protocol.
  See TCP/IP
Try clauses, 186
Types, 76

U

Unary, 97
Unicode, 73
Uniform Resource Locator. *See URLs*
URLs, 364
User input, 52
User interface
  component class, 231
  layout manager, 271
  menus, 229

V

Variable declarations, 35
Variables
  constructors, 37
  declarations, 79
  modifiers, 149
  naming, 36

static, 149
variables, 148
vs. types, 76
Virtual machine, 6, 210
Volatile modifiers, 150

W

Web sites
Coriolis, 25
Javasoft, 54
While, 108
While loops, 210
WhoIs protocol, 349
Widening, 151
Width parameter, 27
Wild cards
hiding classes, 177
Windows, 32

Y

Yield( ) method, 221